Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería del Software

Desarrollo de SMART.CORE mediante arquitecturas REST

Mario Afonso Barreira

Enero 2017

informatika fakultatea
facultad de informática

Resumen

El proyecto consiste en trasladar (reescribir) la tecnología que posee la empresa IDS Ingeniería de Sistemas, S.A. (IDS a partir de ahora) a un modelo basado en servicios, utilizando arquitectura REST. Debido al compromiso de IDS con el software de Microsoft, el marco de trabajo escogido, y en el que deberá ser desarrollado el proyecto, será ASP.NET Web API, el software de Microsoft para el desarrollo de servicios HTTP basados en arquitectura REST.

Teniendo en cuenta la estrategia de producto de IDS, este núcleo deberá ser validado y consumido por un Interfaz de Usuario mínimo (funcionando sobre un “Smartphone” con sistema operativo Android) creado utilizando Xamarin, un entorno ubicado bajo el IDE de Visual Studio especializado en crear aplicaciones "cross-platform” para dispositivos móviles.

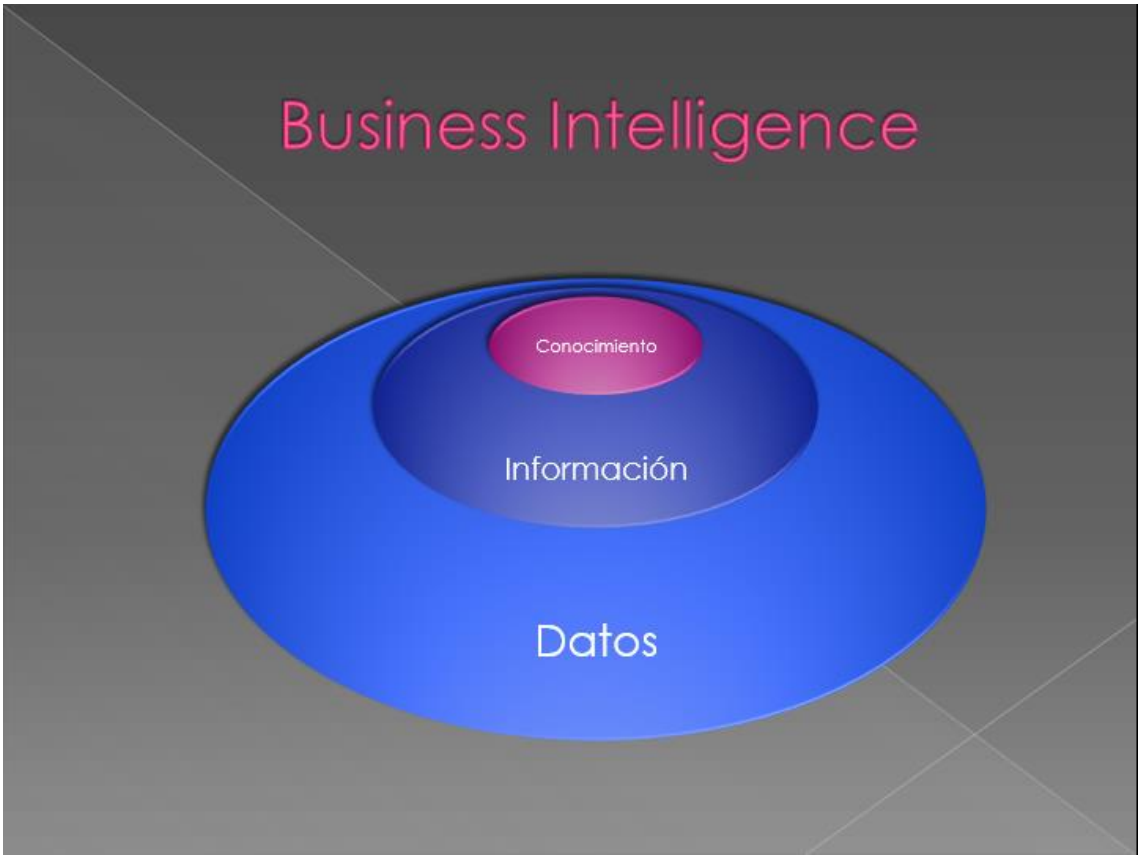
ÍNDICE

1 INTRODUCCIÓN

La tecnología "Smart" es un software que se ubica en el ámbito de la Inteligencia de Negocio (BI - Business Intelligence en su denominación original), un concepto que complementa distintos tipos de aplicativos, principalmente del mundo de los Sistemas de Planificación Empresariales (ERP - Enterprise Resource Planning en su denominación original).

Este tipo de software aborda la manipulación de datos por parte de entidades visuales que convierten a éstos en información. Esta información, posteriormente, es analizada y consumida por usuarios, convirtiéndola, a su vez, en conocimiento aplicable.

El gráfico mostrado a continuación, permite hacerse una idea más clara de este concepto.



El proyecto abordará pues, dos de los tres aspectos fundamentales del "Business Intelligence":

- La extracción de datos desde las fuentes adecuadas.
- La presentación de la información en dispositivos con alta capacidad de visualización.

Debido a la evolución del mercado, en la que se manifiesta una tendencia clara hacia dispositivos de diferente naturaleza (PC, Smartphones, Tabletas, etc.) y que corren software en marcos de ejecución de diversa índole (Web, iOS, Android, Windows, etc.), se ha decidido migrar dicha tecnología "Smart" a un modelo basado en una arquitectura de servicios de tipo REST, de modo que éstos sean

consumibles por diferentes plataformas. Esta nueva tecnología se denominará “eSmart” y el núcleo de la misma y objetivo de este proyecto, “Smart.core”.

Dado el compromiso existente con el software de Microsoft, el marco de trabajo escogido, y en el que deberá ser desarrollado el proyecto, será ASP.NET Web API, el software de Microsoft para el desarrollo de servicios HTTP basados en arquitectura REST.

Teniendo en cuenta la estrategia de producto actual, este núcleo deberá ser validado y consumido por un UI mínimo (funcionando sobre un dispositivo “smartphone” con sistema operativo Android) creado utilizando Xamarin, un entorno ubicado bajo el IDE de Visual Studio especializado en crear aplicaciones "cross-platform" para dispositivos móviles.

2 ANTECEDENTES

IDS Ingeniería de Sistemas, S.A. es una empresa dedicada al desarrollo, implantación y mantenimiento de software de gestión para empresas (ERP, CRM, SGA, BI, Web Sites, etc.). La compañía cuenta con varias aplicaciones modulares con las que satisface las distintas necesidades de este mercado.

Actualmente, se dispone de una tecnología denominada "Smart" que, a grandes rasgos, permite transformar datos ubicados en distintos orígenes (Bases de Datos, Hojas Excel, Archivos de Texto, etc.) en información visible y manipulable (aunque no editable).

Esta tecnología está disponible (directamente o embebida) en la mayoría de las aplicaciones que se desarrollan y comercializan:

- Vector SmartBusiness
- Vector WebSites (a través de SmartWeb)
- Vector CRM (a través de SmartWeb)
- Vector ERP (a través de Vector SmartBusiness for App)

La tecnología "Smart" está basada en un modelo clásico de tres capas más una (Presentación, Negocio, Datos y Sistema) y bajo el marco definido por Microsoft .NET Framework. El lenguaje utilizado es C#.

Esta tecnología se apoya en archivos de formato XML que definen la estructura de lo que, tradicionalmente, se conoce como un "dashboard" o cuadro de mandos, dentro del cual se pueden definir páginas que, a su vez, contienen entidades.

Dichas entidades pueden ser de diferentes tipos:

- Selectores
- Consultas
- Gráficos
- Indicadores
- Fichas
- Otros

Estas entidades se enlazan entre sí, estableciéndose un mecanismo de interdependencia entre ellas, creando un panel heterogéneo de comportamiento dinámico.

En las imágenes que se muestran a continuación, se puede observar el aspecto que tienen, actualmente, los "dashboards" en las diferentes aplicaciones.

- En Vector SmartBusiness:

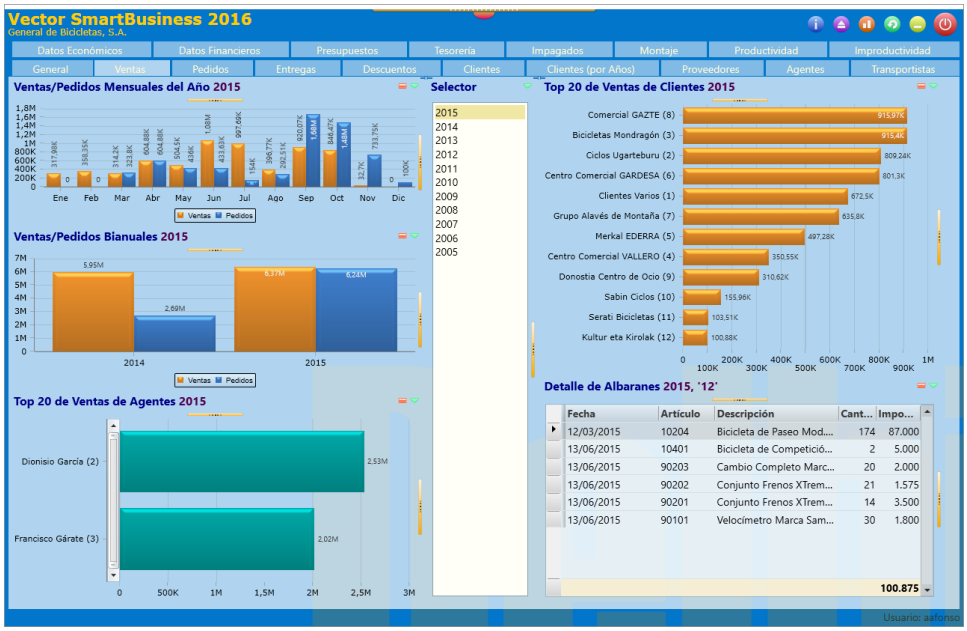


Imagen 1.1 Captura pantalla de SmartBusiness

- En Vector CRM:

Imagen 1.2 Captura pantalla de Vector CRM

- En Vector ERP:

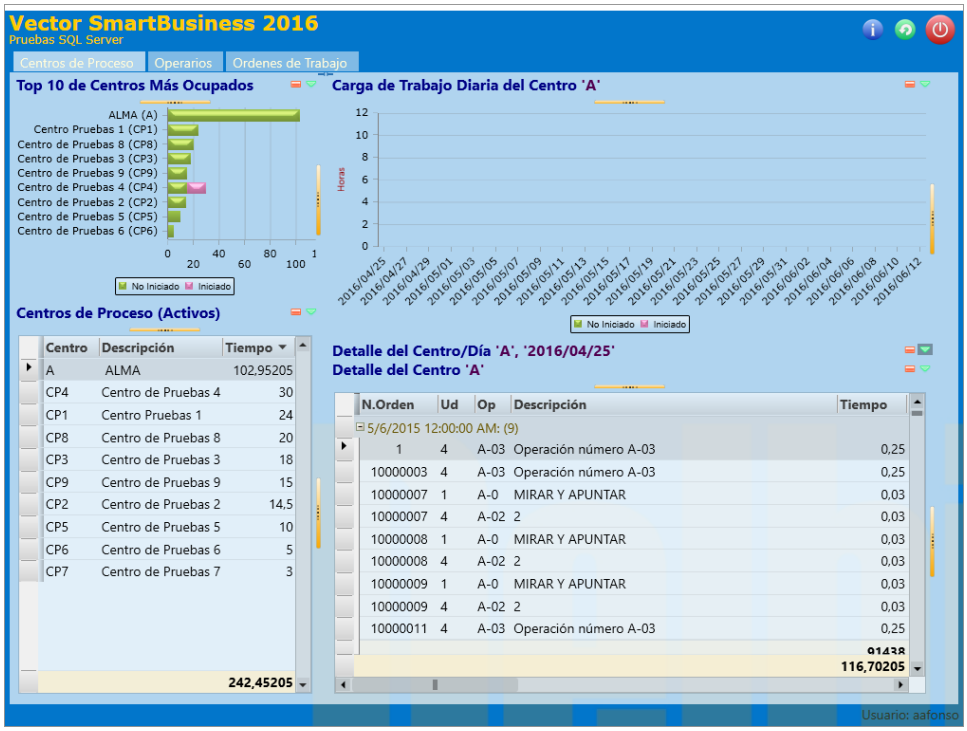


Imagen 1.3 Captura pantalla de SmartBusiness

- En Vector WebSites:

Imagen 1.4 Captura pantalla de Vector WebSites

3 DOCUMENTO DE OBJETIVOS DEL PROYECTO

En este punto, se describirán los principales objetivos del proyecto, los motivos que me han llevado a escogerlo, así como los diferentes factores de riesgo y un análisis de la planificación del proyecto dividido en diferentes fases.

3.1 DESCRIPCIÓN DEL PRODUCTO

El resultado principal de este proyecto, llamado “eSmart.core”, es un núcleo de servicios, con una UI mínima en la que implementaremos la funcionalidad que permita transformar en información visual los datos suministrados por dicho núcleo.

Los objetivos de este proyecto concretamente son:

- Generalizar la tecnología "Smart" actual de IDS (utilizada en los productos Vector SmartBusiness, Vector CRM, Vector WebSites y Vector ERP) a un modelo de servicios consumible desde cualquier UI (eSmart).
- Desarrollar la aplicación en Android que consuma los servicios creados.
- Probar la aplicación con los casos de uso definidos.

3.2 ALCANCE DEL PROYECTO

El alcance del proyecto detalla que tareas se van a realizar y cuales no a lo largo del desarrollo del proyecto.

3.2.1 Informe del alcance

Aunque la mayor parte de este proyecto, será el traslado de la tecnología Smart a eSmart, se desarrollará un pequeño interfaz de usuario en un dispositivo Android que consuma algunos de los servicios web implementados.

En este proyecto se va a realizar una UI móvil mediante la que se podrá:

- Ofrecer a los usuarios una aplicación con la que podrán, consultar visualmente en formato gráfico o tabular, los datos de sus bases de datos.

Las tareas adicionales serán negociables según su viabilidad, una vez finalizado el proyecto, tendrán que estar disponibles los siguientes entregables:

- Aplicación móvil, con una UI básica en la que se podrán consumir los servicios creados.
- Memoria del proyecto terminada.
- Presentación del proyecto terminada.

Los objetivos del proyecto son:

- Desarrollar correctamente el traslado de las funciones a servicios web consumibles, y probar que funcionan correctamente.
- Acabar y entregar la memoria antes de la defensa.
- Realizar la defensa en la convocatoria de febrero.

3.2.2 Conocimientos y actividades de apoyo al alcance del proyecto

Para llevar a cabo este proyecto de manera satisfactoria, son necesarios varios conocimientos de distintas áreas de la Informática, entre ellas, las más destacadas son las técnicas de programación y la gestión del proyecto.

Igualmente, son necesarios conocimientos sobre las bases de datos, principalmente adquiridos en la asignatura Bases de Datos de 2º curso, y ampliados en la asignatura de Diseño de Bases de Datos de 3er curso. También será necesario el dominio del entorno de desarrollo Visual Studio de Microsoft, aprendido en la asignatura de Herramientas Avanzadas del Software, así como del lenguaje de programación C# aprendido en la misma asignatura. También se usarán conocimientos sobre la integración de Visual Studio con las bases de datos, tratados en dicha asignatura.

Debido al tamaño del proyecto, será muy necesario tener una buena planificación de las diferentes fases del proyecto, llevar un riguroso control sobre las reuniones con los directores, tanto de la empresa como el director de mi proyecto, y sobre el desvió entre las horas programadas y las estimadas en un principio.

Además de los conocimientos mencionados anteriormente será muy útil una buena documentación sobre:

- Microsoft Visual Studio .NET
- Xamarin
- Microsoft SQL Server Managment Studio
- Internet Information Services (IIS)

Para llevar un control del proyecto y resolver las dudas que surjan se mantendrán reuniones periódicas con el director del proyecto José Ángel Vadillo, y con Edorta Gisasola, director del proyecto en la empresa IDS.

3.2.3 Plan de dirección del alcance

Debido a que se trata de un Proyecto de Fin de Grado, el alcance del proyecto puede cambiar por varias razones. A continuación, se enumeran los motivos que pueden causar algún desvió en el alcance del proyecto.

- El alumno Mario Afonso, realiza un cambio en el alcance del proyecto, ya sea por iniciativa propia o por fuerza mayor, en caso de que alguna tarea no se pueda realizar por falta de tiempo o conocimiento.
- El tutor del proyecto José Ángel Vadillo, recomienda que se realice algún cambio en el alcance del proyecto.

Una vez acordados los cambios tanto como con el director del proyecto en IDS como con mi director de proyecto José Ángel Vadillo, se procederá a modificar la documentación, teniendo en cuenta las tareas afectadas.

3.2.4 Tareas del equipo del proyecto

Como en todo proyecto, se identificarán los miembros que van a formar parte del equipo. En este apartado, se podrán ver las tareas y obligaciones de cada uno de los miembros del equipo.

- Director del Proyecto- José Ángel Vadillo:
    - La dirección del proyecto será la encargada de:

        - Dirigir y orientar al alumno en la realización del proyecto.
        - Realizar revisiones periódicas del progreso del proyecto.
        - Dirigir y orientar al alumno en el desarrollo de la memoria.

- Director del Proyecto (IDS)- Edorta Gisasola
    - La dirección del proyecto por parte de la empresa, será la encargada de:
        - Impartir formación sobre la tecnología "Smart" actual de IDS. (Nivel usuario).
        - Impartir formación sobre la tecnología "Smart" actual de IDS. (Nivel desarrollador).
        - Dar nociones al alumno sobre cómo se quiere hacer el traslado de la tecnología "Smart" a "eSmart".

- Desarrollador del proyecto – Mario Afonso
    - El desarrollador del proyecto será el encargado de:
        - Realizar las tareas de programación que le sean encomendadas por el director de la empresa.
        - Realizar las reuniones con el director del proyecto para resolver dudas y mantenerle informado sobre el estado del proyecto.

3.2.5 Comunicación entre los miembros del equipo

La realización del proyecto no sería posible sin una buena vía de comunicación entre los integrantes del equipo. Para la comunicación entre el director de proyecto José Ángel Vadillo, el director del proyecto en IDS y el alumno, se utilizarán los siguientes medios:

- Correo electrónico: El correo electrónico en una vía de comunicación formal y oficial, que será utilizada tanto como el alumno con su director de proyecto, el alumno con el director de proyecto en IDS, y ambos directores si fuera necesario. Además, en el correo se permiten intercambio de archivos, por lo que lo convierte en un medio ideal.
- Teléfono: Para dudas puntuales, entre el director del proyecto en IDS y el alumno se utilizará el teléfono o en su defecto, la aplicación de mensajería WhatsApp.
- Reuniones: las reuniones serán de vital importancia en el desarrollo del proyecto, se realizarán reuniones periódicas, tanto como el alumno con el director del proyecto, el alumno con el director de proyecto en IDS, y las tres partes reunidas. Las reuniones constaran de un acta, y se realizaran en el despacho de José Ángel Vadillo o en el despacho del director de Proyecto en IDS.

3.3 DIRECCIÓN DE RIESGOS

Todos los proyectos presentan una serie de actividades peligrosas o factores de riesgo que obligan a tomar medidas cautelares. En caso de que se presentase uno de estos riesgos, el proyecto podría verse afectado, y, por lo tanto, no cumplirse alguno de los objetivos. Para ello, es de vital importancia hacer un análisis de los riesgos, identificándolos y evaluando las posibles medidas cautelares, para que estos no se den, o sean del menor impacto posible en el proyecto.

3.3.1 Identificación de los riesgos

- *Una estimación mal hecha*: una estimación mal hecha, es cuando dedicamos un tiempo excesivo a una tarea, esto es, que nos pasamos del límite que teníamos planificado, y ese tiempo no puede ser recuperado en las siguientes semanas sin tener que cambiar la planificación. *Solución*: Cuando se dé un caso similar, lo que debemos de hacer es cambiar en la planificación, el tiempo de finalización de dicha tarea, o las horas que han de invertirse en ella, todos estos cambios deberán de constatar en la documentación del proyecto.
- *Cambios en los requisitos*: En el caso de que alguno de los directores vea necesario un cambio en alguno de los puntos del proyecto. *Solución*: se realizará una reunión o las que hagan falta con ambos directores hasta llegar a un acuerdo, esto quedará reflejado en la planificación y en las actas de reuniones.
- *Enfermedad*: En el peor de los casos, que uno de los participantes del equipo sufra una enfermedad o un accidente o este de baja, el tiempo estimado para la realización del proyecto puede verse afectado. *Solución*: Se volverá a planificar el alcance del proyecto y la fecha de finalización.
- *Falta de recursos*: Para la realización de este proyecto, hay una serie de recursos necesarios, como un ordenador con ciertos programas instalados, documentación en papel, un móvil con sistema operativo especifico… si alguno de estos fallase, podría verse afectado el tiempo de finalización del proyecto, incluso el alcance del mismo. *Solución*: Se hará lo mismo que en los puntos anteriores, volver a planificar, y en última instancia, una reunión para valorar el no incluir dicho recurso en el proyecto.
- *Dificultad*: Debido a que alguna de las partes del proyecto presente una gran dificultad, la cual superarla, consumiría demasiado tiempo, o fuese inviable, habría que tomar decisiones sobre el alcance del proyecto. *Solución*: para ello, se hará una reunión con ambos directores, y se tomarán las decisiones pertinentes, estas se verán reflejadas en la documentación del proyecto, como en las actas.
- *Inexperiencia del alumno*: Debido a la inexperiencia del alumno en proyectos de esta envergadura, puede que las estimaciones y la planificación realizada al principio, puede que no se ajuste a la real, o que los objetivos del proyecto sean demasiado exigentes. *Solución*: para ello, se hará una reunión con ambos directores, y se tomarán las decisiones pertinentes, estas se verán reflejadas en la documentación del proyecto, como en las actas.
- *Otras actividades complementarias*: teniendo el alumno otras actividades complementarias a las que dedicarle tiempo de estudio, como pueden ser las clases, o los entrenamientos, puede que necesite utilizar tiempo planificado para el proyecto para realizar estas tareas. *Solución*: para ello, se hará una reunión con ambos directores, y se tomarán las decisiones pertinentes, estas se verán reflejadas en la documentación del proyecto, como en las actas.

- *Pérdida de información*: por diferentes motivos, pueden darse perdidas de información, ya sean documentos, ficheros de desarrollo y demás cosas. *Solución*: Realizar copias de seguridad en la nube.

3.3.2 Cuantificación de los riesgos

Cada riesgo identificado anteriormente tiene diferente evaluación, mediante la siguiente tabla se clasifican según la probabilidad de que sucedan y del impacto que generan.

Riesgo	Probabilidad	Impacto	Valoración Riesgo
Estimaciones mal realizadas	Alta	Medio	Medio
Cambios en los requisitos	Media	Alto	Medio
Enfermedad	Baja	Alto	Medio
Falta de recursos	Baja	Alto	Medio
Dificultad	Media	Medio	Medio
Inexperiencia del alumno	Media	Alto	Alto
Otras actividades complementarias	Media	Bajo	Bajo
Pérdida de información	Baja	Alto	Medio

Tabla 3.1. Cuantificación de los riesgos

Después de realizar un análisis, considero que el proyecto es de riesgo Medio.

3.3.3 Análisis de viabilidad

Realizando un análisis de viabilidad, podemos prever el éxito que va a tener el proyecto, y ver si cumple con nuestras expectativas.

Es muy importante, a la hora de emprender un nuevo proyecto, ser realista con los recursos necesarios, y el tiempo estimado, para evitar pérdidas económicas o frustración profesional. Para determinar si este proyecto es viable y así evitar pérdidas, se ha realizado el siguiente estudio:

- *Viabilidad técnica*: Actualmente se dispone de la tecnología necesaria para el desarrollo del proyecto con unas garantías muy prometedoras de satisfacción. Para ello, se dispone de una WorkStation con conexión a internet, copias almacenadas en la nube de la empresa y móviles para las pruebas necesarias.
- *Viabilidad económica*: Dado que este producto no va a lanzarse individualmente al mercado, sino que va a ser una característica añadida a un producto ya existente, su viabilidad económica está tan garantizada como lo esté la del producto al que está vinculado.
- *Viabilidad operativa*: Contamos con los conocimientos y los medios necesarios para poner en marcha la aplicación de forma satisfactoria.
- *Viabilidad legal*: En este punto se valora cualquier posible infracción, violación o ilegalidad que pudiera resultar del desarrollo del proyecto. En principio no hay ningún problema, todas las licencias están en orden y el código es propiedad de la empresa.

Después de estudiar estos puntos, se puede concluir que el proyecto, es viable.

3.4 FASES DEL PROYECTO

Las diferentes fases del proyecto se describen a continuación.

3.4.1 Estudio de las herramientas necesarias

En primer lugar, se realizarán una serie de estudios sobre las herramientas que vamos a utilizar, empezando por el framework de Visual Studio, Xamarin y se estudiarán las distintas posibilidades que ofrece, también recibiré unos cursos de Microsoft SQL server Managment Studio y como se utiliza, para poder trabajar cómodamente con él, por último, recibiré formación sobre Microsoft Internet Information Services (IIS).

3.4.2 Formación en la tecnología "Smart" de IDS

Después de tener conocimiento sobre las herramientas que utilizaremos, será necesario que reciba formación sobre la tecnología "Smart" que usan en la empresa IDS. El encargado de formarme será Edorta Gisasola.

3.4.3 Análisis de requisitos

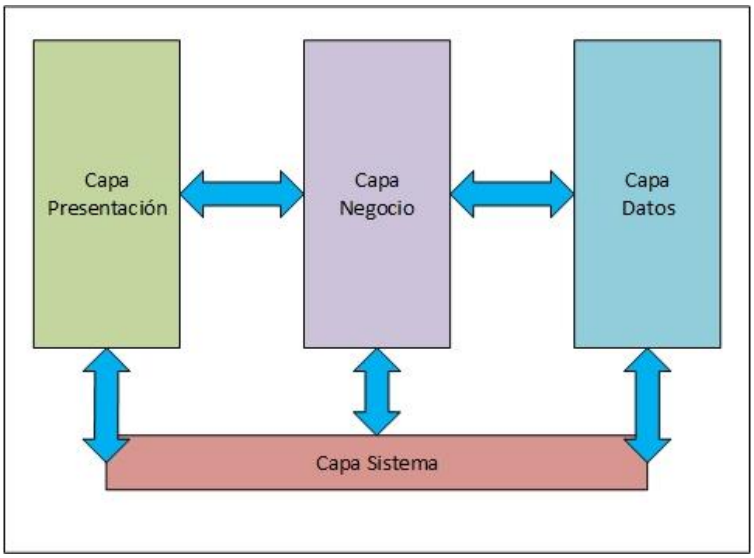
En esta fase se obtendrán los requisitos necesarios para diseñar e implementar el proyecto de forma que se cumplan todos los objetivos. Para ello, durante esta fase, se organizarán reuniones con el director de proyecto de IDS y el director de proyecto José Ángel Vadillo, con el fin de obtener toda la información.

3.4.4 Primera Iteración

En la primera iteración, el objetivo será identificar los requisitos del núcleo REST. Estos serán los mismos, pero re-implementados.

3.4.4.1 Diseño

El diseño del servicio, ya está establecido por la empresa, y su idea es mantener una arquitectura clásica de tres capas más una transversal que podrán utilizar todas. El diseño del servicio seria de la siguiente manera:



3.4.4.2 Implementación

En este punto, después de recibir la formación necesaria, y estar listo para trabajar con el entorno .NET, se procederá a la implementación de las funciones del núcleo "Smart" a servicios web consumibles.

3.4.4.3 Pruebas unitarias de cada funcionalidad

En este punto, lo que haremos será probar individualmente cada una de las funcionalidades que tendra nuestro servicio REST.

3.4.5 Segunda Iteración

En la segunda iteración, el objetivo es diseñar una aplicación móvil en la que poder mostrar una serie de casos de uso a modo de demostración, para que se pueda apreciar el trabajo realizado con el servicio REST.

3.4.5.1 Análisis de requisitos

Sera necesario realizar un análisis de requisitos para nuestra aplicación móvil, y establecer unos actores y unos casos de uso.

3.4.5.2 Diseño de la aplicación

Una vez tengamos los servicios web listos para ser consumidos por una aplicación, procederemos a diseñar la aplicación móvil, utilizando los conocimientos de Xamarin en Visual Studio. Se propondrán varios diseños para la aplicación, desde como deberá funcionar, a que colores usaremos.

3.4.5.3 Implementación

Una vez tengamos el diseño establecido, procederemos a implementar la funcionalidad de esta, haciendo uso de los controles que nos proporciona el API de SyncFusion para Xamarin.

Una vez terminada esta fase, se crearán los manuales de instalación, usuarios y el uso de la aplicación.

3.4.5.4 Pruebas de integración con la primera iteración

Sera necesario ver que todo encaja entre sí, por lo que será conveniente ir probando cada cosa que implementemos, y si funciona con el servicio REST que hemos implementado.

3.4.6 Pruebas y evaluación

La fase de pruebas, servirá para ver los posibles errores de la aplicación, tanto como las mejoras que se le podrían hacer.

Las pruebas serán diseñadas antes de realizarlas. Para comprobar su validez, se harán diferentes pruebas con los dos directores del proyecto, tanto con José Ángel Vadillo como con Edorta Gisasola.

3.4.7 Redacción de la memoria del proyecto

En esta fase, se realizará la redacción de la memoria del proyecto y su revisión.

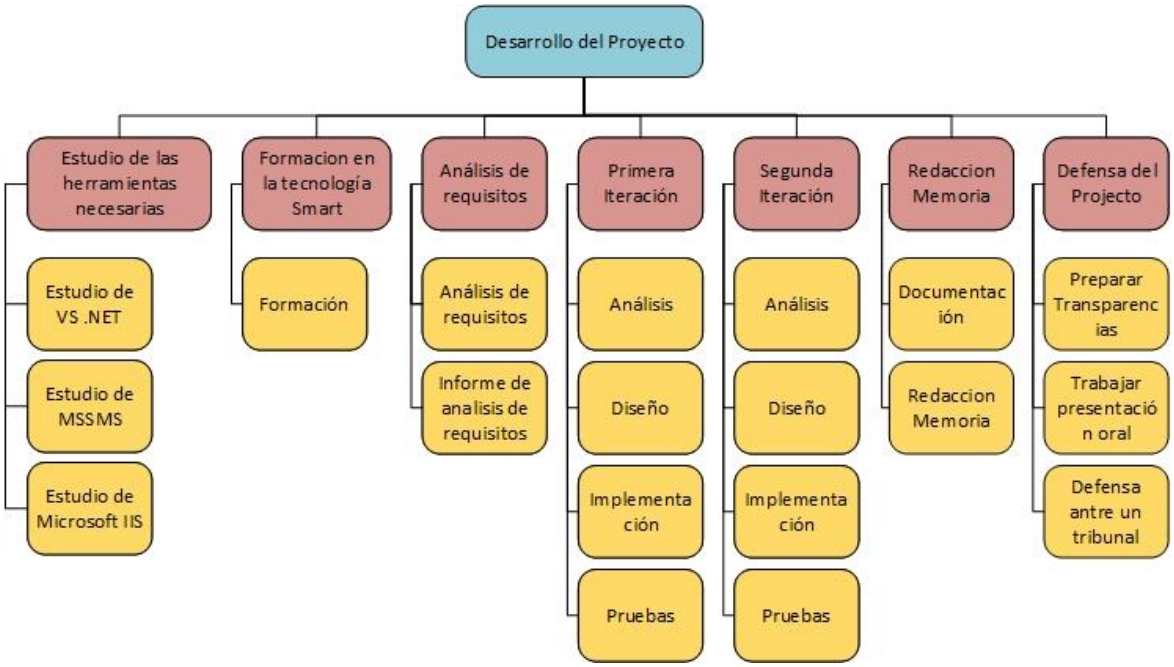
3.4.8 Defensa del proyecto

En esta parte, se preparará la exposición oral del proyecto final ante el tribunal y se considerará la opción de realizar demos en vivo.

3.5 PLANIFICACIÓN DEL PROYECTO

A continuación, se mostrará la planificación realizada para el desarrollo de este proyecto:

3.5.1 Descomposición de Tareas del Proyecto



3.5.2 Diagrama de Gantt

En este diagrama se muestra el orden en el que se realizarán las diferentes tareas planificadas para el proyecto, aporta información sobre la dependencia entre ellas, la paralelización de las tareas y los hitos principales.

Los plazos definidos en este diagrama, tratan de ajustarse a la medida de lo posible al tiempo real invertido en el desarrollo del proyecto, pero en ningún caso pretenden ser una definición fija de los plazos y fechas en los que se llevará a cabo una tarea.

De esta manera, no será más sencillo detectar los desvíos y retrasos en las tareas.

En esta planificación, vemos un modelo de 4 horas diarias los 5 días de la semana, se realizará entre los días 9 de septiembre y 11 de enero, un total de 4 meses, teniendo en cuenta que los fines de semana serán festivos, pudiéndose utilizar estos como comodines para posibles retrasos que surjan durante el desarrollo del proyecto, nos quedarían unos 110 días, en resumen 440 horas.

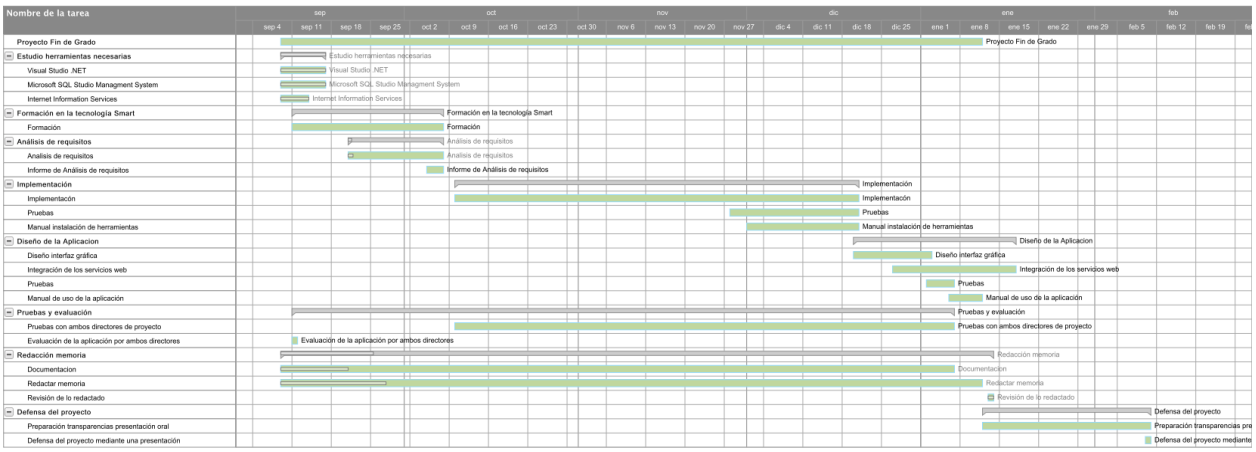


Diagrama Gantt 3.1. Debido a que la imagen no se ve muy bien, dejamos un link al pdf original.

4 ANÁLISIS DE FACTIBILIDAD

Todo este proyecto, se realiza con la intención de sacar un producto comercial y que dé beneficios. Se trata de estandarizar lo máximo posible un modelo que ya poseen, por lo que está sólidamente justificado su desarrollo.

Las nuevas tecnologías como Xamarin, permiten abordar este proyecto desde un punto de vista diferente, en el que trabajar para varias plataformas a la vez, para una sola persona, ya no es una idea tan descabellada.

Además, poseemos una de las herramientas más avanzadas del mercado en cuanto a entorno de trabajo, Microsoft Visual Studio .NET. Y el lenguaje de programación más impulsado por la compañía Microsoft, C#.

En cuanto a la factibilidad del cronograma, se ha hecho una estimación del tiempo necesario para la implementación del proyecto y se ha visto que, sí es posible terminarlo a tiempo, aunque sin mucho margen de error.

4.1 APLICACIONES NATIVAS VS CROSS-COMPILE

Para entender mejor cuál de las dos opciones se adecua más a nuestras necesidades, haremos una comparativa entre las dos opciones.

4.1.1 Nativas

Las aplicaciones nativas se realizan para una plataforma específica, para correr directamente y sólo en esa plataforma. Se usan lenguajes específicos y herramientas diferentes. Así por ejemplo tenemos:

- Objective C / Swift para aplicaciones iOS.
- Java para aplicaciones Android.

4.1.1.1 Pros

Los pros de las aplicaciones nativas son:

- Ofrecen el mejor rendimiento, al no haber capas intermedias.
- Al interactuar directamente con las APIs ofrecidas por cada fabricante, se tiene un mayor control.

4.1.1.2 Contras

Los contras de las aplicaciones nativas son:

- Requiere conocer el lenguaje y entorno de desarrollo específicos de cada plataforma.
- Por ser específicas, no hay reusabilidad de código entre plataformas.

4.1.2 Cross-compile

Este tipo de aplicaciones podrían considerarse híbridas, pero siguen un tipo de estrategia diferente a las de tipo WebView.

En ellas, se realiza una transformación a partir de nuestro código a elementos UI nativos, usando "runtimes" propios, consiguiendo con ello un rendimiento similar a las aplicaciones nativas, pero sin llegar a programar en la plataforma específica. No es un cross-compile pero se acerca.

Aquí tenemos tecnologías como:

- Xamarin: permite desarrollar para dispositivos móviles en C#.
- Titanium Appcelerator: se utiliza JavaScript, con un framework MVC llamado Alloy.

4.1.2.1 Pros

- Permiten reutilizar conocimientos en lenguajes concretos.
- Permiten compartir código entre plataformas, usando sólo un lenguaje. No totalmente, pues dependiendo del producto puede que no se puedan compartir elementos.

4.1.2.2 Contras

- Como en Telerik, estas opciones tienen un coste económico y de aprendizaje.

- Hay menos libertad para desarrollar, ya que estás atado a las herramientas que te ofrece la plataforma.

5 REQUISITOS DEL SISTEMA

El análisis de requisitos en la primera etapa de un proyecto software, en ella se tratan de definir las tareas necesarias a realizar para unos usuarios con tal de resolver un problema.

Objetivos:

- Traducir una serie de funciones del núcleo "Smart" que posee IDS a servicios web, consumibles desde cualquier aplicación web.
- Crear una aplicación móvil con una interfaz sencilla en Android para probar estos servicios.

5.1 REQUISITOS TECNOLÓGICOS Y HERRAMIENTAS

Desde el principio del proyecto, las herramientas principales se han establecido las siguientes:

- Entorno de desarrollo: Microsoft Visual Studio .NET 2015 Comunity.
- Lenguaje de desarrollo: Microsoft C#.
- Servidor: Microsoft Internet Information Services.
- Gestor de la Base de Datos: Microsoft SQL Server Management Studio.

Estas herramientas, no han sido escogidas por mí, sino que han sido impuestas por la empresa IDS, ya que tienen establecidos varios convenios con Microsoft para la utilización de sus productos, y es su método de trabajo.

Una de las ventajas de usar la plataforma .NET es su enorme comunidad. Hay cantidad de foros con solución a la mayoría de los problemas.

5.2 INSTALACIÓN DE PROGRAMAS NECESARIOS

Para llevar a cabo el desarrollo de este proyecto, seran necesarias varias herramientas de desarrollo, las cuales seran descritas en las siguientes lineas.

Visual Studio Comunity 2015 – Esta herramienta sera la principal, la que mas usaremos. Ha sido instalada con una licencia de estudiante, proporcionada por la UPV. Se instalará un Framework, en forma de add-on llamado Xamarin, del cual hablare mas adelante.

Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, Django, etc., a lo cual sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas, etc. Fuente – Wikipedia

Tendremos que instalar el framework de Xamarin para visual studio, este se puede adquirir gratuitamente desde su pagina web.

MSSMS (Microsoft SQL Server Managment Studio) – Microsoft SQL Server es un sistema de manejo de bases de datos del modelo relacional, desarrollado por la empresa Microsoft.

El lenguaje de desarrollo utilizado (por línea de comandos o mediante la interfaz gráfica de Management Studio) es Transact-SQL (TSQL), una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos (DML), crear tablas y definir relaciones entre ellas (DDL).

Dentro de los competidores más destacados de SQL Server están: Oracle, MariaDB, MySQL, PostgreSQL. SQL Server solo está disponible para sistemas operativos Windows de Microsoft. Fuente – Wikipedia

En nuestro proyecto lo usaremos para hacer las modificaciones necesarias en las base de datos con el fin de hacer varias pruebas y poenr diferentes casos.

IIS (Internet Information Services) – Internet Information Services o IIS1 es un servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows. Originalmente era parte del Option Pack para Windows NT. Luego fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, como Windows 2000 o Windows Server 2003. Windows XP Profesional incluye una versión limitada de IIS. Los servicios que ofrece son: FTP, SMTP, NNTP y HTTP/HTTPS.2

Este servicio convierte a un PC en un servidor web para Internet o una intranet, es decir que en los ordenadores que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente.

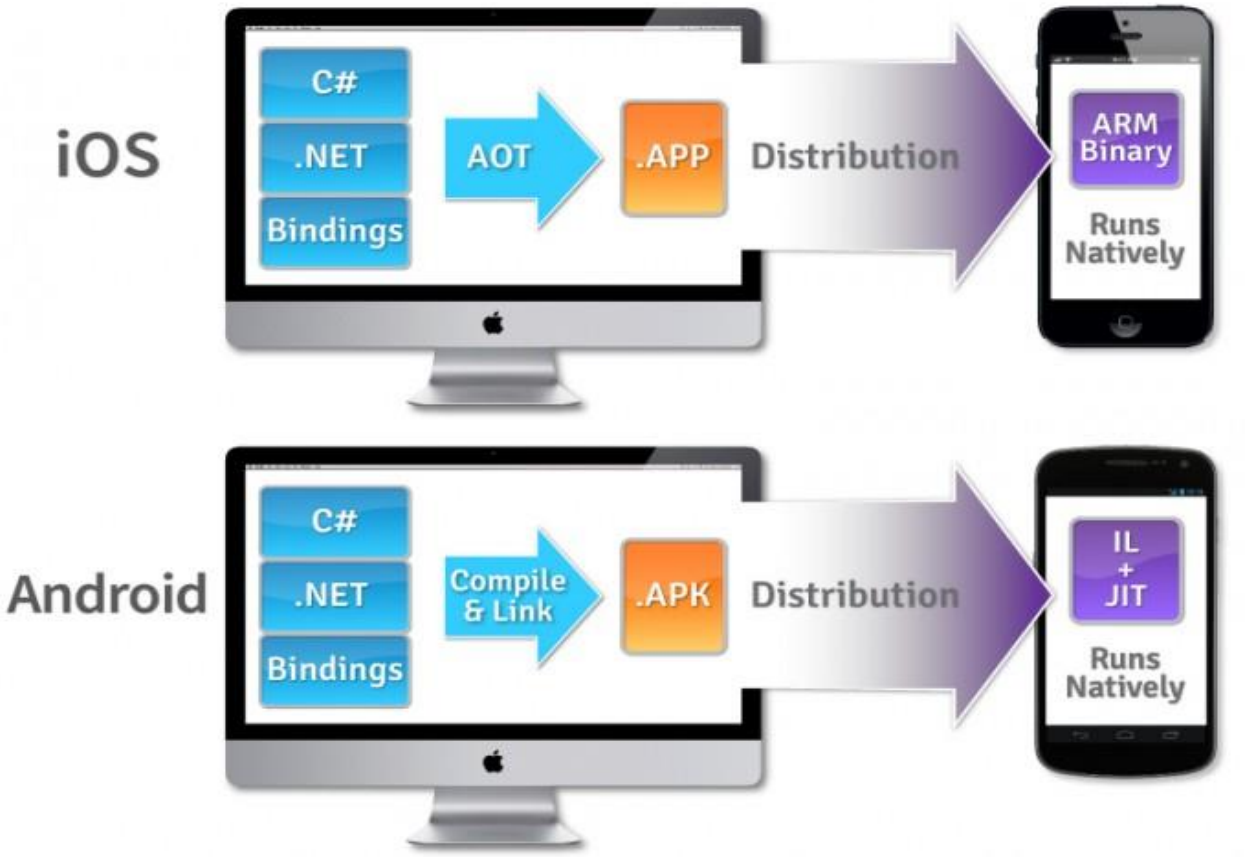
Se basa en varios módulos que le dan capacidad para procesar distintos tipos de páginas. Por ejemplo, Microsoft incluye los de Active Server Pages (ASP) y ASP.NET. También pueden ser incluidos los de otros fabricantes, como PHP3 o Perl.4 .

Fuente – Wikipedia

5.3 ¿QUÉ ES XAMARIN?

Xamarin es una compañía que se estableció en mayo de 2011, recientemente adquirida por Microsoft, y liderada por los mismos ingenieros que crearon el proyecto Mono, consistente en una implementación libre de la plataforma de desarrollo .NET para dispositivos Android, iOS y GNU/Linux. Es decir, con Xamarin podríamos evitar tener que utilizar Java para desarrollar una aplicación para Android. Anteriormente, este proyecto se llamaba MonoTouch y MonoDroid.

El principal motivo para que surjan plataformas de este tipo radica en que uno de los mayores desafíos a la hora de desarrollar aplicaciones multiplataforma, es mantener la consistencia en todos los entornos (en términos de desarrollo en paralelo de todas las plataformas, con unos tiempos y costes razonables).

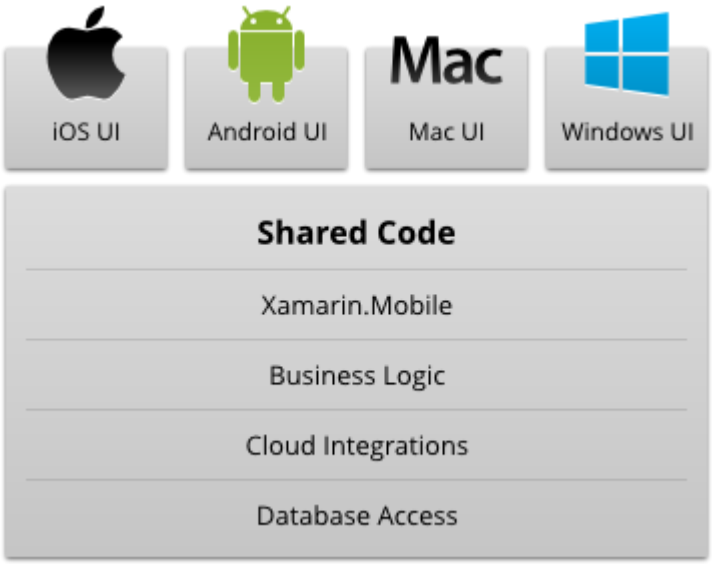


Pero, ¿cuál es la idea que hay detrás de Xamarin? El concepto es sencillo: utilizar C# y .NET para la compilación de aplicaciones nativas para Android, iOS o Windows 10.

5.3.1 Ventajas

Xamarin nos permitirá generar nuestra aplicación para iOS (.APP), para Android (.APK) y para Windows (.UWP), la cual ya sí correrá de forma nativa. Gracias a esto, surge una de las grandes ventajas de Xamarin: la reutilización de código. En cualquier aplicación multiplataforma que hayamos desarrollado, hay módulos iOS que hemos tenido que portar a Java, o módulos Android que hemos tenido que portar a Objective-C. Pero en este caso, al desarrollar todas las plataformas en la misma tecnología, no es necesario reescribir el código, al poder reutilizar módulos ya implementados.

La cosa no queda ahí, pues al poder desarrollar aplicaciones nativas para Windows Phone, Windows 8 y web en C# y .NET, resulta que la reutilización del código la estamos exportando aún a más plataformas que iOS o Android. Según Xamarin, deberíamos poder reutilizar el 90% del código aproximadamente.



Otra de las ventajas de la plataforma es que dispone de tantas librerías de terceros como el código nativo: hay una gran comunidad detrás. Pero uno de los detalles más importantes es que Xamarin nos proporciona acceso total a la API estándar de Android.

Esto nos permite no estar limitados en funcionalidad por el hecho de utilizar Xamarin en lugar de Android nativo. ¿Hasta qué punto se parecen Xamarin y Android nativo? Lo mejor es ver un ejemplo a continuación muy sencillo, donde una actividad carga un layout. Veamos la versión nativa:

```
import android.app.Activity;
import android.os.Bundle;
public class HolaMundoActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

El código es muy sencillo, pero sirve de ejemplo perfecto para el caso. Ahora veamos un código similar, pero en Xamarin:

```
10 namespace Hello_World
11 {
12     [Activity (Label = "Hello_World", MainLauncher = true)]
13     public class Activity1 : Activity
14     {
15         int count = 1;
16 
17         protected override void OnCreate (Bundle bundle)
18         {
19             base.OnCreate (bundle);
20 
21             // Set our view from the "main" layout resource
22             SetContentView (Resource.Layout.Main);
23 
24             // Get our button from the layout resource,
25             // and attach an event to it
26             Button button = FindViewById<Button> (Resource.Id.myButton);
27 
28             button.Click += delegate {
29                 button.Text = string.Format ("{0} clicks!", count++);
30             };
31         }
32     }
33 }
```

Este código hace exactamente lo mismo, pero además añade un "listener" a un botón que se encuentra en el layout principal. ¿No es totalmente similar?

Y esa es una de las ventajas que personalmente me gustaría destacar de Xamarin: aprender Xamarin implica aprender Android implícitamente y viceversa. ¿Por qué? Porque al final, la API que vamos a utilizar va a ser totalmente equivalente, ya sea en Java o en C#/.NET.

Fuente - http://www.elandroidelibre.com/2014/05/xamarin-la-api-para-crear-aplicaciones-multiplataforma-en-c-net.html#

5.3.2 Desventajas

Tras ver las ventajas de Xamarin, sus inconvenientes más destacables son que todo el código no será reutilizable y que reaprovechemos más o menos código dependerá de nuestras capacidades de desacoplar funcionalidad, pues todo el código relacionado con manejo de interfaces gráficas no será reutilizable. Por eso, es nuestra misión extraer de forma totalmente independiente toda la funcionalidad posible. La pega es que, si nuestra aplicación tiene mucho código relacionado con la interfaz gráfica, menos será el código que tenemos que reutilizar. Y es cierto que toda la lógica para la interfaz gráfica (gestión de eventos, "listeners" …) al final resulta ser bastante código.

Otro inconveniente es que cualquier aplicación en Xamarin ocupa más espacio que una nativa, afectando al tiempo de descarga y al almacenamiento. Hablamos de que mínimo ocupará unos cuantos Mb y puede crecer proporcionalmente con el código si éste utiliza más características de la API. Esto se debe a cómo se ensambla el código desde .NET para enlazarlo a código nativo. Esto también provoca, en el caso de Android, un retardo inicial al abrir la aplicación para el caso de Xamarin. Es cierto que la compañía trabaja en reducir todo esto y está consiguiendo avances, pero aún es remarcable.

La comunidad de Xamarin sigue creciendo… Pero, ¿es comparable a lo que mueven compañías como Google (Android), Apple (iOS)…? Claro que no. La comunidad de Xamarin, comparada con las grandes, es pequeña, y eso puede provocar también que nos cueste más trabajo encontrar algo específico o que directamente no esté.

Otro de los inconvenientes de Xamarin, puede verse también como una ventaja. Antes comentábamos que aprender Xamarin implica aprender Android, por el tema de gestionar las interfaces gráficas. Pero esto nos lleva a que hacer una aplicación multiplataforma con Xamarin, al final sabremos cómo funciona la API de Android y la API de iOS.

Aunque en nuestro proyecto, todo esto se ve disminuido por el hecho de que tenemos la oportunidad de utilizar componentes de Syncfusion, e implementarlo todo en formato xaml, y el propio compilador se encargara de traducir toda la interfaz gráfica de nuestro proyecto. Esto repercutirá en una menor personalización de nuestra interfaz, pero nos dará una rapidez de programación y seguridad de que funcionará por igual en todas las plataformas.

Fuente - http://www.elandroidelibre.com/2014/05/xamarin-la-api-para-crear-aplicaciones-multiplataforma-en-c-net.html#

5.3.3 Conclusión

Esta plataforma de desarrollo, se adapta perfectamente a la empresa. Debido a que en los programas que quieren trasladar a plataformas móviles, solo será necesario el diseño de la capa de presentación, mientras que todo el trabajo, lo realizará un servicio desde Azure, por lo que se ahorrará dinero y tiempo al crear las aplicaciones.

Y debido a que solo tendremos que programar una vez la interfaz gráfica, con alguna pequeña excepción, esta es la mejor opción para nuestro proyecto.

5.4 REQUISITOS DE RECURSOS

En este apartado se enumeran los recursos tanto de hardware como de software que serán utilizados durante el proyecto.

Hardware

El componente principal para el desarrollo de este proyecto es un PC y los requisitos mínimos para poder correr el software fluidamente son los siguientes:

	Requisitos mínimos	Estación de trabajo
Procesador	Procesador de 1,6 GHz o superior	Intel I5-6500 3,6GHz
Memoria RAM	1 GB de RAM (1,5 GB si se ejecuta en una máquina virtual)	16GB de RAM DDR4
Espacio Disco Duro	10 GB de espacio disponible en el disco duro	256GB de espacio en disco duro
Tipo Disco	Unidad de disco duro de 5400 rpm	Unidad de disco de estado sólido (SSD)
Tarjeta video	Tarjeta de vídeo compatible con DirectX 9 con una resolución de pantalla de 1024 x 768 o superior	Nvidia GTX 960ti 4GB DDR5
Monitor	Monitor con una resolución superior a 800x600 con 256 colores de alta resolución	Monitor HP 23" FULLHD 1920x1080

Tabla 5.1. Tabla de requisitos hardware

También será necesario un dispositivo Android, para poder probar la interfaz gráfica. En este caso, disponemos de un terminal proporcionado por la empresa, en concreto un LG G3, actualizado a la última versión de Android, la 7.0, que cuenta con 2Gb de memoria RAM, 32Gb de almacenamiento y una pantalla de 5,5" Quad HD (2560 x 1440 px).

Software

El entorno será Microsoft Visual Studio .NET 2015 Comunity, la escogida por la empresa, y en la cual tengo experiencia de asignaturas ya cursadas. Para que la aplicación se pueda correr será necesario cumplir los siguientes requisitos:

- El servidor web debe ser el Internet Information Service (IIS) con la versión 10.0 o superior
- Un emulador de Android con versión superior a la 4.4 KitKat. Aunque la app estará optimizada para funcionar en Android 5.0 Lollipop.

Para la gestión con la base de datos de la empresa, será necesario usar el Microsoft SQL Service Management Studio 2016.

5.5 REQUISITOS FUNCIONALES

En este apartado se explicará de forma detallada los pasos que han de realizar los diferentes tipos de usuarios dentro de la aplicación:

Tendremos dos aplicaciones, una basada en servicios web con unos requisitos funcionales predefinidos, y una aplicación móvil (Android) en la que a nivel de usuario se deben establecer unos casos de uso. Mi tarea será la adaptación de su tecnología a un servicio REST y la creación de la aplicación ya mencionada.

6 DESARROLLO DEL SISTEMA

Tendremos dos sistemas, uno será el servicio REST, y el otro la aplicación cliente. Ambas siguen una estructura de 3 niveles, más una capa llamada “Sistema”, que es común a los 3 niveles.

6.1 ARQUITECTURA DEL SISTEMA

En este punto, trataremos de mostrar las principales diferencias entre los dos apartados principales del proyecto, el servicio REST y la aplicación móvil.

6.1.1 Servicio REST

Los niveles de la aplicación se dividen en 4, el primer nivel es la capa de presentación, el segundo la lógica de negocio, el tercero la capa de datos y el cuarto, es un nivel común a todos, y es la capa “Sistema”.

6.1.1.1 Capa de Presentación

El proyecto llamado “eSmartCoreService”, es el que se encarga de los controladores y llamar a las funciones necesarias de la capa de negocio con los datos que se le proporcionen desde la aplicación cliente.

6.1.1.2 Lógica de negocio

El segundo nivel es la capa de negocio, en ella tenemos todas las funciones que se encargan de transformar los datos, básicamente es donde está la funcionalidad de la aplicación.

6.1.1.3 Capa de datos

El tercer nivel es la capa de datos, aquí es donde se accede a la base de datos y están las funciones que tengan que ver con ella.

6.1.1.4 Capa de sistema

El cuarto nivel es la capa de sistema, en esta se encuentran los objetos que se van a usar en la aplicación, en este caso llamados “WParts”, la definición de una página “Smart” y del dashboard.

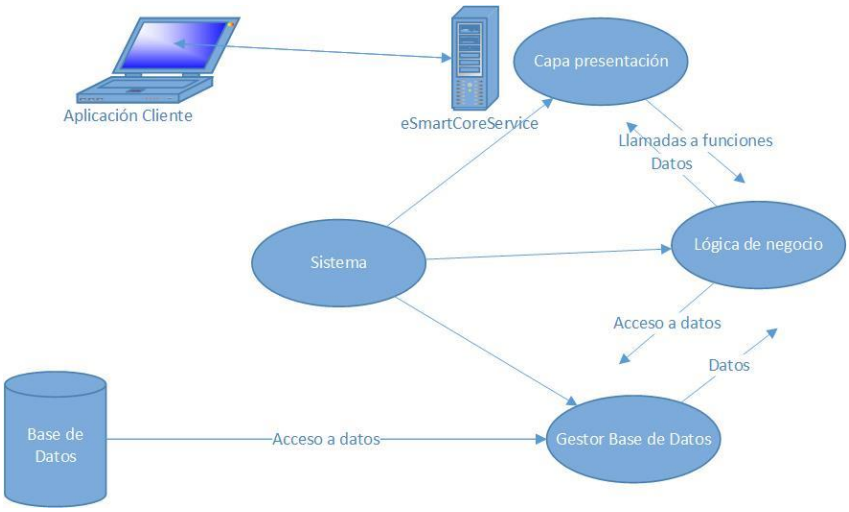


Imagen 6.1. Arquitectura de 4 niveles, servicio REST

6.1.2 Aplicación Cliente

La aplicación que desarrollaremos como demo para probar el servicio REST, también se divide en 4 niveles, al igual que el propio servicio REST. Capa de presentación, de negocio, de datos y una común a todas, de sistema.

6.1.2.1 Capa de presentación

Aquí podemos encontrar el diseño de la aplicación en Xaml, como estamos desarrollando un proyecto cross-platform con Xamarin, esta plantilla Xaml será común al código de Android, iOS, y Windows Phone. También nos permite desarrollar la vista Xaml en código C#, que es donde haremos las llamadas a las funciones. Esta capa de presentación, constara de 6 proyectos.

- eSmartCoreClient (Portable)
- eSmartCoreClient.Droid
- eSmartCoreClient.iOS
- eSmartCoreClient.UWP (Universal Windows)
- eSmartCoreClient.Windows (Windows 8.1)
- eSmartCoreClient.WinPhone (Windows Phone 8.1)

Aunque nosotros solo editamos el código del primer proyecto, luego a la hora de compilar para cada una de las diferentes plataformas, Xamarin se encargará de traducir el código correspondiente.

6.1.2.2 Lógica de negocio

En la lógica de negocio de esta aplicación, tendremos las funciones que serán necesarias para hacer que la aplicación muestre los datos correctamente y, en general, toda la funcionalidad de la aplicación.

6.1.2.3 Capa de datos

Esta es la capa que se encarga de obtener los datos necesarios, en este caso, no llamando a una base de datos, sino al servicio REST, que será el encargado de proporcionarle los datos.

6.1.2.4 Capa de sistema

Esta capa es común a la del servicio REST, ya que comparten las mismas definiciones de objetos.

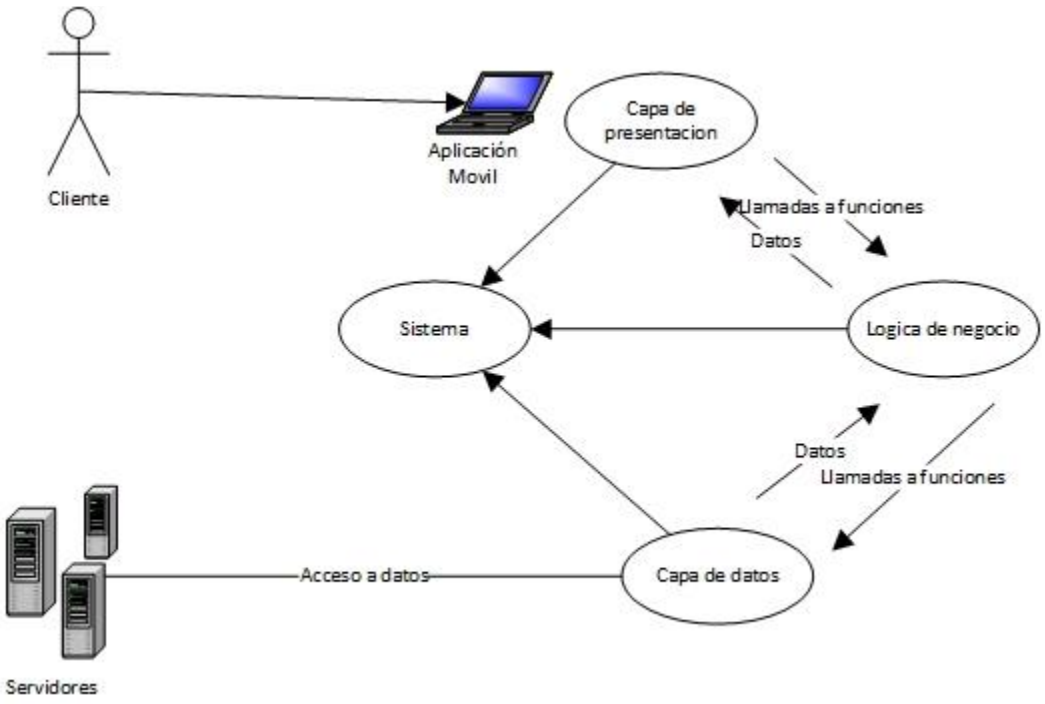


Imagen 6.2. Arquitectura de 4 niveles, aplicación cliente

6.2 DESARROLLO DEL NÚCLEO REST

En este punto, explicaremos detalladamente el diseño que teníamos, cómo lo hemos adaptado a nuestras necesidades, en que consiste, y como han sido implementadas las funcionalidades.

6.2.1 Análisis

Como ya se ha indicado en apartados anteriores, la empresa necesita una modernización de su tecnología, y mi trabajo es implementar un servicio REST para que sea consumido por cualquiera de sus aplicaciones que utiliza la tecnología "Smart".

En primer lugar, se ha de realizar el análisis de requisitos funcionales de la aplicación.

Para este proyecto, se decidió que la aplicación debería contener todas las definiciones de los objetos que utiliza la empresa IDS, pero que, de momento, solo tendría que implementar la funcionalidad de unos casos concretos.

Dichos casos son:

- Petición de una lista de las paginas disponibles en la base de datos
- Petición de una entidad particular:
    - Un gráfico
    - Una consulta de tipo lista
    - Una consulta de tipo ficha
- Petición de los datos de un grafico
- Petición de los datos de una consulta de tipo lista
- Petición de los datos de una consulta de tipo ficha

Al tener que usar protocolos de la empresa, la seguridad e integridad de los datos corre a su cargo. Pero cabe decir que todo esto estará alojado en un servidor en la nube, y que contará con toda la seguridad que puede proporcionar el servicio Microsoft Azure.

Una vez definidas claramente las funcionalidades que debe tener, podemos pasar al diseño de éstas.

6.2.2 Diseño

El diseño del servicio REST y sus elementos, viene ya predefinido por la empresa, por lo que en este apartado no ha sido necesario que realizase nada. Simplemente tenía que ocuparme de implementar correctamente los diseños que ya existían.

6.2.2.1 Diagramas de Secuencia

A continuación, mostraremos los diagramas de secuencia de los casos que se han implementado, y los cuales están definidos en la parte del cliente. En los diagramas que se ven en la parte del cliente, se muestra el funcionamiento en la aplicación móvil, pero la parte del servicio REST está representada por un dibujo, el cual se explicara aquí.

6.2.2.1.1 Petición de la lista de Paginas Smart

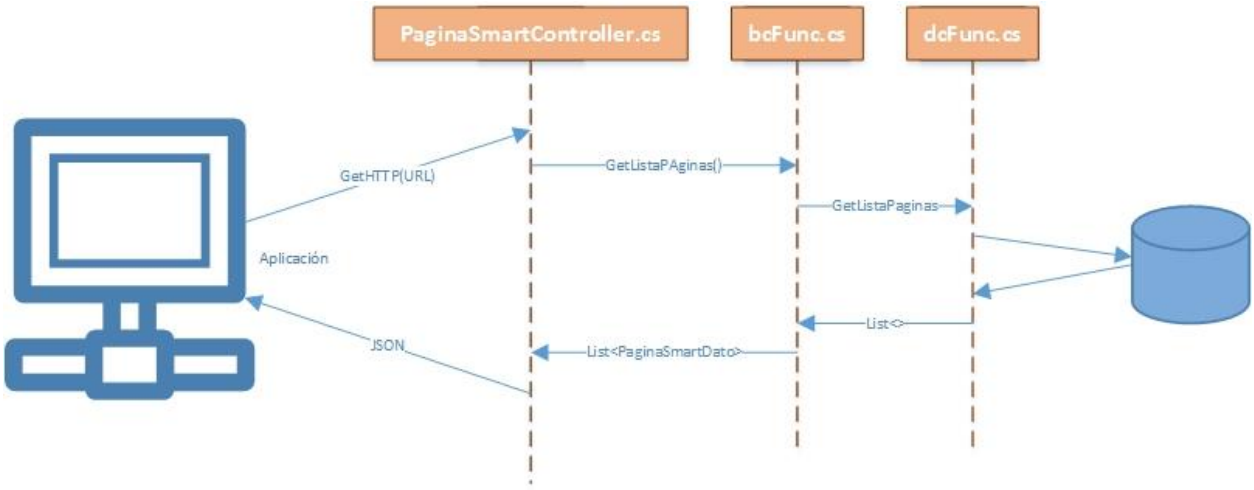


Imagen 6.3. Ver imagen en tamaño completo, pág. 143.

Desde el cliente recibimos una petición HTTP para obtener la lista de páginas disponibles.

El acceso a la base de datos se realiza mediante Entity Framework y el resultado se envía en formato JSON.

6.2.2.1.2 Petición de una Pagina Smart

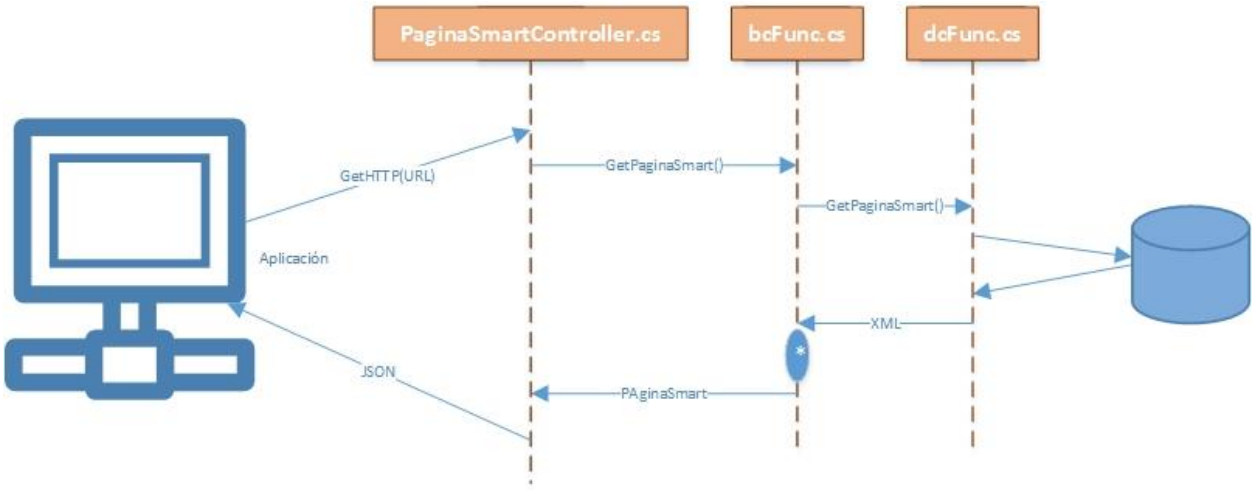


Imagen 6.4. Ver imagen en tamaño completo, pág. 144.

Desde el cliente recibimos una petición HTTP para obtener la configuración de una página “Smart”. Al recibir el código de página, con ello, el acceso a la base de datos se realiza mediante Entity Framework y obtenemos el registro concreto de la página. La configuración obtenida de la base de datos está en formato XML que lo transformaremos a objetos y ese resultado se envía serializado en formato JSON.

6.2.2.1.3 Petición de datos de un grafico

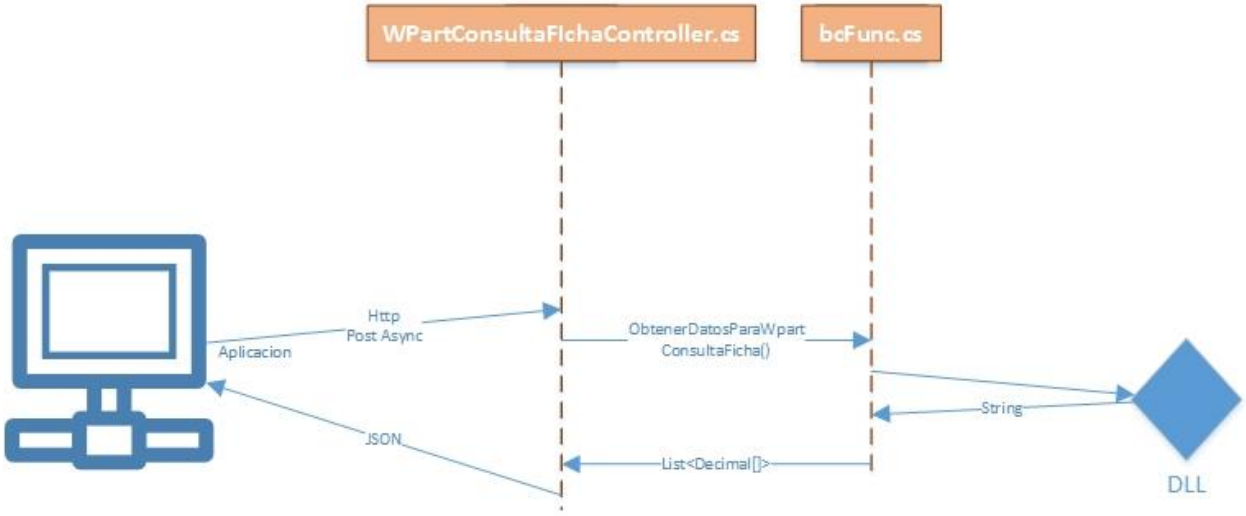


Imagen 6.5. Ver imagen en tamaño completo, pág. 145.

Desde el cliente recibimos una petición HTTP donde se trasfiere la configuración del gráfico y con ella mediante la utilización de una DLL suministrado por la empresa devolvemos los datos en formato JSON.

6.2.2.1.4 Petición de datos de una consulta

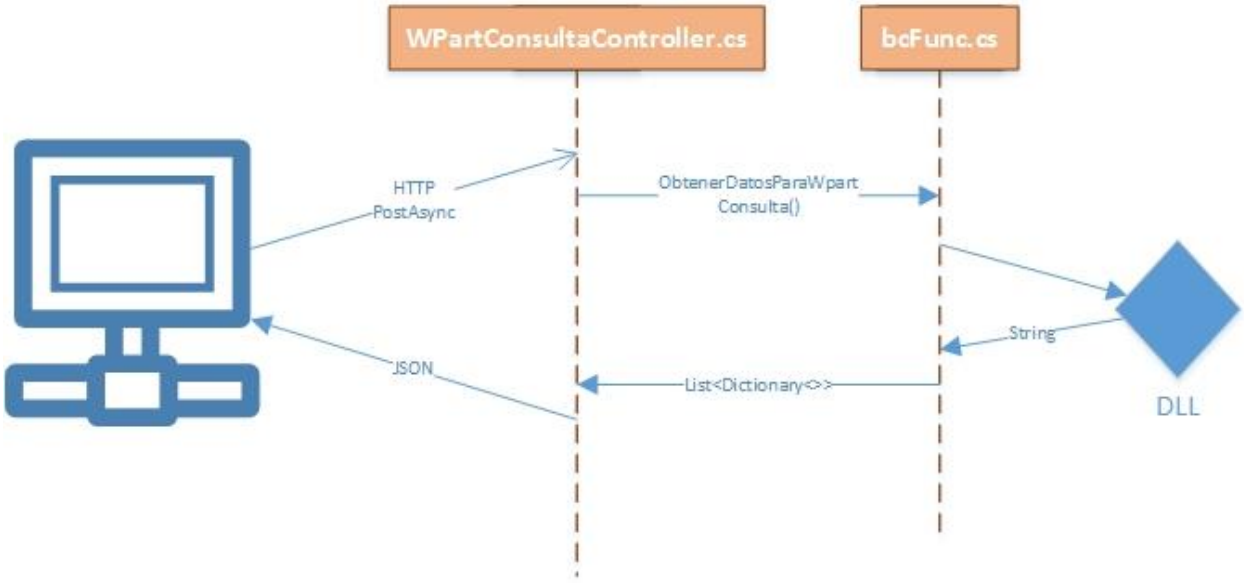


Imagen 6.6. Ver imagen en tamaño completo, pág. 146.

Desde el cliente recibimos una petición HTTP donde se trasfiere la configuración de la consulta y con ella mediante la utilización de una DLL suministrado por la empresa devolvemos los datos en formato JSON.

6.2.2.1.5 Petición de datos de una consulta ficha

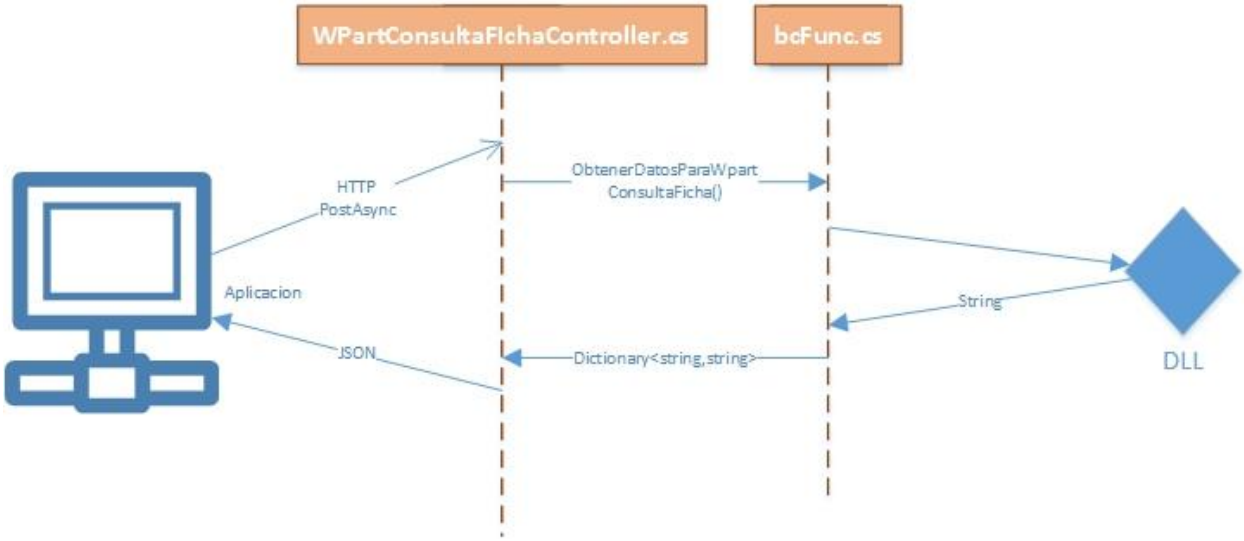


Imagen 6.7. Ver imagen en tamaño completo, pág. 147.

Desde el cliente recibimos una petición HTTP donde se trasfiere la configuración de la consulta ficha y con ella mediante la utilización de una DLL suministrado por la empresa devolvemos los datos en formato JSON.

6.2.3 Implementación

En este apartado veremos una detallada explicación de la implementación que he hecho de la tecnología "Smart" que tienen en la empresa. Obviamente, esto es simplemente una pequeña parte de todo lo que abarca "Smart", pero suficiente para este proyecto.

A continuación, procederé a mostrar como he implementado cada una de los diagramas de secuencia detallados en el punto de diseño. Cabe destacar que la aplicación REST tiene un apartado destacado que sería la capa de sistema que además es común a la aplicación cliente. Por ello, he considerado necesario explicarlo como un punto importante dentro de este apartado.

6.2.3.1 Capa sistema

El diseño del sistema (que es común a ambas partes, tanto cliente como servicio), contiene los objetos que usaran ambas partes (Servicio y Cliente). En ella se encuentran la definición de la página "Smart", de las "WPart" y el resto de clases auxiliares.

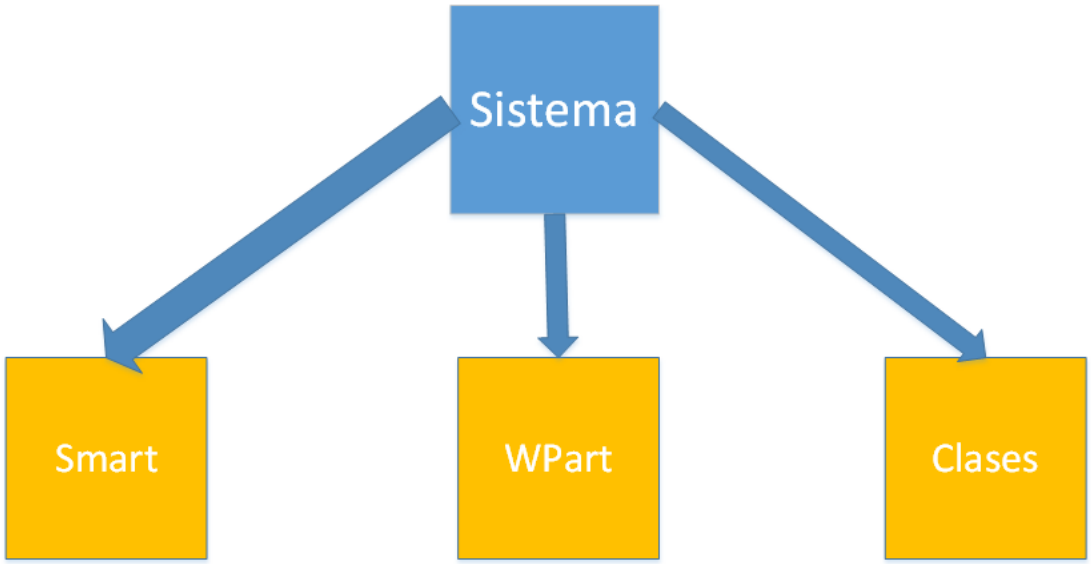


Imagen 6.8. Estructura capa sistema.

6.2.3.1.1 WParts

Las entidades "WPart" son la definición de la configuración de cada componente visual individual que se representará en el cliente. Con estos componentes visuales, el cliente es capaz de mostrar información en distintos formatos, como, gráficos, calendarios, etc. Todos ellos heredan de una clase general, que es "WPart" y luego cada una extiende su clase con las propiedades específicas.

En todos estos objetos encontramos dos apartados, el primero con la definición de las variables privadas y el segundo con las propiedades públicas. Las variables privadas sólo son utilizadas en el ámbito de la propia clase y se definen con un guion bajo por delante. Las propiedades públicas sirven para leer o escribir los valores que contienen las variables privadas desde fuera del objeto, cada una

de las propiedades tienen definida las funciones get y set. Las propiedades públicas se llaman igual que las privadas, pero sin el guion bajo.

A continuación, se detalla cada una de ellas explicando el uso de sus variables. Muchas de estas clases contienen más variables definidas que las que explico. Esto se debe a que la configuración que tenemos en la base de datos en formato XML es utilizada desde otros programas más complejos y que en este proyecto no ha hecho falta utilizarlos.

6.2.3.1.1.1 WPart

```
public class WPart
    {
        #region Variables Privadas

        protected string _Clase = "";
        protected string _Nombre = "";
        protected string _Titulo = "";
        protected int _Columna = 0;
        protected int _Linea = 0;
        protected int _SpanLinea = 0;
        protected int _SpanColumna = 0;
        protected string _FondoVentana = "Transparente";
        protected List<Comando> _Comandos = new List<Comando>();
        protected string _InstruccionesComandos = "";
        protected int _Indice = 0;
```

Objeto 6.1. Wpart.

- `Clase`: contiene el tipo de WPart que es
- `Nombre`: contiene el identificador único de la WPart en la página Smart
- `Titulo`: contiene el texto descriptivo que aparecerá en la página Smart
- `FondoVentana`: es el color que tiene la WPart de fondo
- `Indice`: es el orden de la WPart en la página Smart

6.2.3.1.1.2 WPartGrafico

Este es el primero de los tres que podremos ver en ejecución en la demo de la aplicación. Este `WPart` es el que se encarga de contener la configuración visual de los gráficos.

```
public class WPartGrafico : WPart
{
      #region Variables Privadas

      protected string _TituloEjeX = "";
      protected string _NumeroValoresEjeX = "";
      protected string _FormatoEtiquetasEjeX = "";
      protected string _TituloEjeY = "";
      protected string _ValorMaximoEjeY = "";
      protected string _FormatoEtiquetasEjeY = "";
      protected string _TipoOrdenacion = "";
      protected string _Opciones = "";
      protected List<SerieWPartGrafico> _ListaSeries = new
      List<SerieWPartGrafico> ();
```

Objeto 6.1.1. WpartGrafico.

- `TituloEjeX`: contiene el texto del eje X
- `NumeroValoresEjeX`: se puede limitar a una cantidad de números para visualizar el resultado
- `FormatoEtiquetasEjeX`: definición del formato de las categorías del eje x
- `TituloEjeY`: contiene el texto del eje y
- `ValorMaximoEjeY`: significa que se puede limitar el valor máximo a visualizar
- `FormatoEtiquetasEjeY`: definición del formato de los textos del eje y
- `TipoOrdenacion`: ascendente, descendente
- `Opciones`: Se pueden definir la cantidad de decimales a mostrar en los valores, etc.

Y una lista de series, en la que irán guardados los diferentes gráficos, ya que en un `WPartGrafico` puede albergar más de uno. Su estructura es la siguiente:

```
public class SerieWPartGrafico
        {
            #region Variables Privadas

            protected string _NombreSerie = "";
            protected string _TituloSerie = "";
            protected string _NombreTabla = "";
            protected string _EjeX = "";
            protected string _EjeY = "";
            protected string _Filtro = "";
            protected string _TipoGrafico = "";
            protected string _Funcion = "";
            protected string _Opciones = "";
            protected string _ColorGrafico = "";
```

Aquí podemos ver los datos de cada gráfico, que consta de:

- `NombreSerie`: El nombre de la serie
- `TituloSerie`: El título de la serie
- `NombreTabla`: Nombre de la tabla de la que se extraen los datos
- `EjeX`: Valores del eje X, el campo en la tabla que representa el valor de la categoría

- `EjeY`: Valores del eje Y, el campo en la tabla que representa el valor numérico
- `Filtro`: El filtro que se va a aplicar, puede estar configurado previamente para una preselección de datos
- `TipoGrafico`: El tipo de grafico que es, de barras, líneas, áreas, donut, tarta, etc.
- `Funcion`: Las funciones que se le aplican al gráfico, se le pueden aplicar funciones a los valores de tipo, media, porcentaje, etc.
- `Opciones`: Las opciones que tiene
- `ColorGrafico`: El color del grafico

Color del grafico

Los colores de los gráficos están diseñados por la empresa, esta es la gama de colores que utilizan:

De izquierda a derecha, los colores son:

```
{"Rojo","#D2413C"},
{"Naranja","#F4952D"},
{"Ambar","#F6C500"},
{"Amarillo","#F6E100"},
{"Pistacho","#9EC64A"},
{"Verde","#559C27"},
{"Turquesa","#00ABAA"},
{"Cyan","#30E5E0"},
{"Azul","#4081D1"},
{"Indigo","#6676E1"},
{"Lila","#7470FF"},
{"Violeta","#7A58A5"},
{"Magenta","#F35FBC"},
{"Rosa","#F08DC8"},
{"Blanco","#FFFFFF"},
{"Gris","#A3A3A3"},
{"Grafito","#383838"},
{"Negro","#000000"}
```

6.2.3.1.1.3 WPartConsulta

Este es el segundo `WPart` que podremos ver en la aplicación de demostración. Este consiste en mostrarnos una consulta, la cual tiene la siguiente estructura:

```
public class WPartConsulta : WPart
{
	#region Variables privadas

	protected string _NombreTabla = "";
	protected List<string> _Campos = new List<string>();
	protected List<ConfiguracionCampoConsulta> _TitulosCampos = new
	List<ConfiguracionCampoConsulta>();
	protected List<string> _CamposSubtotales = new List<string>();
	protected List<string> _CamposAgrupacion = new List<string>();
	protected List<string> _CamposOrdenacion = new List<string>();
	protected string _CamposDeAgregado = "";
	protected string _CamposDeFiltrado = "";
	protected string _Filtro = "";
	protected string _Opciones = "";
	protected string _DatoABuscar = "";
	protected string _Condiciones = "";

	protected List<string> _ListaCtrEnlaceSalida = new List<string>();
	protected List<string> _CamposEnlace = new List<string>();
```

Objeto 6.1.2. WpartConsulta

Sus variables, representan lo siguiente:

- `NombreTabla`: El nombre de la tabla del que se extraen los datos
- `Campos`: Una lista con los campos de los valores de la consulta
- `TitulosCampos`: Una lista de tipo `ConfiguracionCampoConsulta` que se encarga de guardar una serie de propiedades de un campo de consulta como el título, el ancho… su estructura es la siguiente:

```
public class ConfiguracionCampoConsulta
{
	protected string _NombreCampo;
	protected string _Titulo; // posición 0
	protected string _Ancho; // posición 1
	protected string _Opciones; // posición 2
```

- `CamposSubtotales`: Una lista con los campos subtotales, son los campos que representan la suma de valores
- `CamposAgrupacion`: Una lista con los campos de agrupación, son los campos con los que se agrupan distintos registros de la consulta
- `CamposOrdenacion`: Una lista con los campos de ordenación, son los campos por los que se ordena la consulta
- `CamposDeAgregado`: Los campos de agregado, son campos no pertenecientes a la tabla al que se le aplican funciones
- `CamposDeFiltrado`: Los campos de filtrado, son campos en los que la consulta parece filtrada visualmente
- `Filtro`: Los filtros que se le aplicaran al `WPart`, en el momento de extraer los registros de la base de datos

- `Opciones`: Las opciones que tendrá el `WPart`
- `Condiciones`: Las condiciones, no se han implementado por su complejidad y representa la configuración de colores de letra dependiendo del valor del registro, etc.
- `ListaCtrEnlaceSalida`: Una lista de los controles de enlace de salida, estos son los WParts que cuelgan de él, en la demostración, podremos ver que hay WParts de tipo `ConsultaFicha` que dependen de él.
- `CamposEnlace`: Una lista con los campos de enlace, aquí es donde se configura el campo de esta `WPart` que le pasara el valor al campo de la `WPart` dependiente

6.2.3.1.1.4 WPartConsultaFicha

Este es el último `WPart` que podremos ver implementado en la aplicación de demostración. Es un elemento que normalmente se usa para ver los detalles de algo en concreto, en la demo, principalmente veremos que se usa para ver los datos de un registro en concreto.

Como todos los `WParts` hereda de `WPart` todos sus atributos, y añada los suyos propios:

```
public class WPartConsultaFicha : WPart
{
      #region Variables Privadas

      protected string _NombreTabla = "";
      protected List<string> _Campos = new List<string>();
      protected List<ConfiguracionCampoConsultaFicha> _TitulosCampos = new
      List<ConfiguracionCampoConsultaFicha>();
      protected string _Filtro = "";
      protected string _Opciones = "";
```

Objeto 6.1.3. WpartConsultaFicha

Podemos ver que sus variables son:

- `NombreTabla`: El nombre de la tabla del que se extrae el registro
- `Campos`: Una lista de los campos que mostraran valores
- `TitulosCampo`: Una lista de tipo `ConfiguracionCampoConsultaFicha`, que es simplemente:

  ```
  public class ConfiguracionCampoConsultaFicha
  {
          protected string _NombreCampo;
          protected string _Titulo;

  }
  ```

  que se encarga de guardar el nombre del campo junto con su título.
- `Filtro`: El filtro que va a llevar
- `Opciones`: Y las opciones que se aplican a la ficha

6.2.3.1.1.5 WPartConsultaINI

Este `WPart` no ha sido implementado en la demo de la aplicación, pero su diseño e implementación están terminados y son correctos. Esta `WPart` se utiliza para representar el valor de un campo en un registro en formato `campo` = `valor`. Su estructura es la siguiente:

```
public class WPartConsultaINI : WPart
    {
      #region Variables Privadas
      protected string _NombreTabla = "";
      protected string _CampoMemo = "";
      protected string _ColumnaCampo = "";
      protected string _ColumnaTipo = "";
      protected string _ColumnaOrden = "";
      protected string _Filtro = "";
      protected string _Opciones = "";
```

Objeto 6.1.4. WpartConsultaINI

Contamos con los siguientes parámetros:

- `NombreTabla`: El nombre de la tabla del que se extraen los datos
- `CampoMemo`: Campo memo, es el nombre del campo donde se encuentra el valor en formato INI
- `ColumnaCampo`: El campo en la tabla donde está definido el campo
- `ColumnaTipo`: Es el campo en la tabla donde está definido de que tipo es el campo
- `ColumnaOrden`: Es el campo en la tabla donde se define el orden de aparición de los campos
- `Filtro`: El filtro que se aplicara al `WPart`
- `Opciones`: Las opciones que tendrá esta `WPart`

6.2.3.1.1.6 WPartGantt

Este WPart no ha sido implementado en la aplicación de demostración, pero su diseño e implementación están terminados y son correctos. Como su propio nombre indica, este WPart es el descriptor de un diagrama de Gantt. Y su estructura es la siguiente:

```
public class WPartGantt : WPart
    {
      #region Variables Privadas

      protected string _NombreTabla = "";
      protected string _CampoId = "";
      protected string _CampoParentId = "";
      protected string _TituloCampoId = "";
      protected string _CamposDescriptor = "";
      protected string _TituloCamposDescriptor = "";
      protected string _CampoFechaInicio = "";
      protected string _TituloCampoFechaInicio = "";
      protected string _CampoHoraInicio = "";
      protected string _CampoFechaFin = "";
      protected string _TituloCampoFechaFin = "";
      protected string _CampoGradoAvance = "";
      protected string _CampoHoraFin = "";
      protected string _Vista = "";
      protected string _Filtro = "";
      protected string _Opciones = "";
      protected string _Horizonte = "";
```

Objeto 6.1.5. WpartGantt

Aquí podemos ver las siguientes variables:

- `NombreTabla` El nombre de la tabla del que se extraen los datos
- `CampoId`: El campo de identificación
- `CampoParentId`: El campo de identificación del padre
- `TituloCampoId`: El título del campo de identificación
- `CamposDescriptor`: Campos descriptor, es la descripción del identificador
- `TituloCamposDescriptor`: El título del campo descriptor
- `CampoFechaInicio`: Un campo con la fecha de inicio
- `TituloCampoFechaInicio`: El título del campo fecha de inicio
- `CampoHoraInicio`: Un campo con la hora inicial
- `CampoFechaFin`: Un campo con la fecha final
- `TituloCampoFechaFin`: El título del campo fecha final
- `CampoGradoAvance`: Un campo con el grado de avance
- `CampoHoraFin`: Un campo con la hora final
- `Vista`: La vista que representa inicialmente el diagrama, diaria, mensual o anual
- `Filtro`: El filtro que se aplica al `WPart`
- `Opciones`: Las opciones que lleva el `WPart`
- `Horizonte`: El horizonte, los datos que se visualizan inicialmente entre unas fechas

6.2.3.1.1.7 WPartIndicador

Este `WPart` no ha sido implementado en la aplicación de demostración, pero su diseño e implementación están terminados y son correctos. El indicador se compone de varias series que

muestran un único valor numérico, representándolo en distintos formatos como tacómetros, termómetros, etc. Y su estructura es la siguiente:

```
public class WPartIndicador : WPart
{
      #region Variables Privadas

      protected string _Opciones = "";
      protected List<SerieWPartIndicador> _ListaSeries = new
      List<SerieWPartIndicador>();
```

Objeto 6.1.6. WpartIndicador

Sus variables representan lo siguiente:

- `Opciones`: Las opciones del indicador
- `ListaSeries`:

```
public class SerieWPartIndicador
{
      #region Variables Privadas

      protected string _NombreSerie = "";
      protected string _TituloSerie = "";
      protected string _NombreTabla = "";
      protected string _Campo = "";
      protected string _Filtro = "";
      protected string _ValoresLimiteIndicador = "";
      protected string _Escala = "";
      protected string _TipoIndicador = "";
      protected string _DireccionIndicador = "";
      protected string _Funcion = "";
      protected string _Opciones = "";
      protected string _ColorIndicador = "";
```

  Una lista con las series de los indicadores del tipo `SerieWPartIndicador` en la que podemos ver:
  - `NombreSerie`: Un nombre de la serie
  - `TituloSerie`: Un título de la serie
  - `NombreTabla`: Nombre de la tabla de la que se extraen los datos
  - `Campo`: Campo en la tabla que se extrae el valor
  - `Filtro`: El filtro que se le aplica al WPart
  - `ValoresLimiteIndicador`: Los valores límite del indicador. Máximo, mínimo y puntos medios
  - `Escala`: El valor se indica en una escala
  - `TipoIndicador`: El tipo de indicador que es, tacómetro, termómetro, etc.
  - `DireccionIndicador`: La dirección del indicador, ascendente, descendente.
  - `Funcion`: Función, se pueden aplicar porcentajes, medias, etc.
  - `Opciones`: Las opciones que tendrá el indicador
  - `ColorIndicador`: Y el color del indicador.

Color del indicador

Los colores de los indicadores están diseñados por la empresa, esta es la gama de colores que utilizan:

De izquierda a derecha, los colores son:

```
{"Rojo","#FF680500"},
{"Naranja","#FF9D2D00"},
{"Ambar","#FF9D5000"},
{"Amarillo","#FFE8C200"},
{"Pistacho","#FF5F7D1B"},
{"Verde","#FF005908"},
{"Turquesa","#FF24826B"},
{"Cyan","#FF008A8C"},
{"Azul","#FF1A5A8C"},
{"Indigo","#FF000563"},
{"Lila","#FF474063"},
{"Violeta","#FF380263"},
{"Magenta","#FF560963"},
{"Rosa","#FF65386E"},
{"Marfil","#FFE0D4C1"},
{"Gris","#FF666666"},
{"Grafito","#FF282828"},
{"Negro", "#FF000000"}
```

6.2.3.1.1.8 WPartMultiselector

Este WPart no ha sido implementado en la aplicación de demostración, pero su diseño e implementación están terminados y son correctos. Esta `WPart` sirve para enlazar distintos `WPart` donde se realiza el filtro para preseleccionar datos. Y su estructura es la siguiente:

```
public class WPartMultiselector : WPart
{
      #region Variables Privadas

      protected string _Opciones = "";
      protected string _ListaCtrEnlaceSalida = "";
      protected List<SerieWPartMultiselector> _ListaSeries = new
      List<SerieWPartMultiselector>();
```

Objeto 6.1.7. WpartMultiselector

Sus variables hacen referencia a lo siguiente:

- `Opciones`: Las opciones con las que se mostrara el `WPart`
- `ListaCtrEnlaceSalida`: Una lista con los controles de enlace de salida, ya que de este `WPart` dependerán varios. Por lo que necesitamos saber cuáles son.

```
public class SerieWPartMultiselector
{
        #region Variables Privadas

      protected string _NombreSerie = "";
      protected string _TituloSerie = "";
      protected string _CamposSerie = "";
      protected string _TipoControl = "";
      protected string _TipoSelector = "";
      protected string _NombreTabla = "";
      protected string _CampoValor = "";
      protected string _CampoDescriptor = "";
      protected string _Filtro = "";
      protected string _RangoValores = "";
      protected string _Opciones = "";
      protected List<ValorPorDefecto> _ValorPorDefecto = new
      List<ValorPorDefecto>();
      protected string _CriterioValidacion = "";
      protected string _Condiciones = "";
```

Y una lista de los `selectores` que habrá, de tipo `SerieWPartMultiselector`:
Y cada selector constara de:

- `NombreSerie`: Un nombre de la serie
- `TituloSerie`: Un título de la serie
- `CamposSerie`: Los campos que tendrá la serie
- `TipoControl`: El tipo de control, puede ser un selector de calendario, radio button, combo-box, etc.
- `TipoSelector`: El tipo de selector, es donde se representa si es una fecha, un texto, etc.
- `NombreTabla`: El nombre de la tabla del que se extraen los datos para mostrarlos en el selector
- `CampoValor`: El nombre del campo que se extrae el valor
- `CampoDescriptor`: El nombre del campo que se extrae la descripción
- `Filtro`: El filtro que se le aplicará a la hora de extraer los datos de la tabla
- `RangoValores`: El rango, en el caso de querer representar una lista de valores concretos
- `Opciones`: Las opciones con las que se mostrará
- `ValorPorDefecto`: Una lista de valores por defecto del tipo `ValorPorDefecto`, cuya estructura es:

```
public class ValorPorDefecto
{
      protected string _Campo = "";
      protected string _Valor = "";
```

- `CriterioValidacion`: Un criterio de validación

6.2.3.1.1.9 WPartScheduler

Este `WPart` no ha sido implementado en la aplicación de demostración, pero su diseño e implementación están terminados y son correctos. Es una vista de calendario, donde se visualizan los registros ubicados por fecha. Y su estructura es la siguiente:

```
public class WPartScheduler : WPart
{
  #region Variables Privadas

  protected string _NombreTabla = "";
  protected string _CampoId = "";
  protected string _TituloCampoId = "";
  protected string _CamposDescriptor = "";
  protected string _TituloCamposDescriptor = "";
  protected string _CampoFechaInicio = "";
  protected string _CampoHoraInicio = "";
  protected string _TituloCampoFechaInicio = "";
  protected string _CampoFechaFin = "";
  protected string _CampoHoraFin = "";
  protected string _TituloCampoFechaFin = "";
  protected string _Vista = "";
  protected string _Filtro = "";
  protected string _Opciones = "";
```

Objeto 6.1.8. WPartScheduler

Lo que estas variables representan en el calendario es:

- `NombreTabla`: El nombre de la tabla del que se extraen los datos
- `CampoId`: Su identificador
- `TituloCampoId`: El título del campo identificador
- `CamposDescriptor`: Un campo descriptor, que describe el registro en el calendario
- `TitulosCamposDescriptor`: El título de los campos descriptores
- `CampoFechaInicio`: Un campo con la fecha de inicio
- `CampoHoraInicio`: Un campo con la hora de inicio
- `CampoFechaInicio`: El título del campo fecha de inicio
- `CampoFechaFin`: Un campo con la fecha final
- `CampoHoraFin`: Un campo con la hora final
- `TituloCampoFechaFin`: El título del campo fecha final
- `Vista`: La vista que representa inicialmente el calendario, diaria, mensual, anual o agenda
- `Filtro`: El filtro que se le aplicará al calendario
- `Opciones`: Las opciones con las que se mostrará el calendario.

6.2.3.1.2 Smart

Aquí encontraremos las dos principales clases de objetos con los que se define esta tecnología. Como he comentado anteriormente, los `WPart` están contenidos en páginas "Smart" y a su vez estas pueden estar agrupadas en un `Dashboard`.

6.2.3.1.2.1 Dashboard

La clase `Dashboard` es una definición para agrupar páginas `Smart`. Su definición es muy sencilla y tendrá un título y una lista de páginas `Smart`.

```
namespace CSistema.Smart
{
    public class Dashboard
    {
        protected string _titulo = "";
        protected List<PaginaSmart> _Paginas = new List<PaginaSmart>();
    }
}
```

Objeto 6.2. Dashboard.

Vemos que consta de:

- `Título`: El título del dashboard
- `Paginas`: Una lista de páginas Smart

6.2.3.1.2.2 PaginaSmart

La clase `PaginaSmart` es la definición de cómo se agrupan las `WPart`, en ella tendremos el código de la `página`, `titulo`, `ratio`, `columnas`, `filas` y `entidades` (los `WParts` de los que consta).

```
public class PaginaSmart
{
      protected string _CodigoPagina = "";
      protected string _Titulo = "";
      protected string _Ratio = "";
      protected List<DefinicionColumna> _Columnas = new
      List<DefinicionColumna>();
      protected List<DefinicionFila> _Filas = new List<DefinicionFila>();
      protected List<WPart> _Entidades = new List<WPart>();
```

Objeto 6.3. PaginaSmart.

Vemos que consta de:

- `CodigoPagina`: El código de la página, este es su identificador
- `Título`: El título de la pagina

- `Columnas`: Una lista de definiciones de columnas del tipo `DefinicionColumna`, cuya estructura es la siguiente:

```
public class DefinicionColumna
    {
        protected string _Ancho = "";
```

- `Filas`: Una lista de definiciones de filas del tipo `DefinicionFila`, cuya estructura es la siguiente:

```
public class DefinicionFila
    {
        protected string _Alto = "";
```

`Entidades`: Y una lista de las diferentes entidades de las que consta, que son los WParts que ya hemos descrito anteriormente.

6.2.3.1.3 Clases

En este apartado encontraremos clases y funciones suministrados por la empresa que he tenido que copiar y adaptar para poder usarlas en la aplicación cliente, puesto que las DLL donde están escritas actualmente no pueden ser usadas por la aplicación Xamarin.

La razón por la que hemos reescrito algunas de las funciones de las librerías que nos proporciona la empresa, es porque los proyectos de tipo Cross-Platform, es necesario el uso de librerías portables. Y en esas librerías no tenemos el "framework" completo.

6.2.3.2 Funcionamiento del servicio REST

En este punto explicaremos como esta implementada las funcionalidades de la aplicación, desde el lado del servicio. En ella mostraré cómo interactúa el sistema desde las capas de presentación a las de datos pasando por la de negocio.

6.2.3.2.1 Petición de la lista de Paginas Smart

Una vez que recibimos la petición de obtener la lista de páginas desde el cliente, tenemos que ir desde el controlador que recibe dicha petición a la capa de negocio.

```
public List<PaginaSmartDato> Get()
{
      bcFunc negocio = new bcFunc();

      return negocio.GetListaPaginas();
}
```

La capa de negocio, se encargará de llamar a la capa de datos.

```
public List<PaginaSmartDato> GetListaPaginas()
{
	dcFunc datos = new dcFunc();

	List<PaginaSmartDato> resultado = datos.GetListaPaginas();

	return resultado;
}
```

Una vez en la capa de datos, mediante la función `GetListaPaginas`() y usando LINQ[1], extraemos de la base de datos, en concreto de la tabla VPART y obtenemos las páginas que están disponibles. Luego, estas son procesadas para que sean objetos del tipo `PaginaSmartDato` y se juntan todas en una lista de nombre "resultado".

```
public List<PaginaSmartDato> GetListaPaginas()
{
	using (WebCRMGEntities context = new WebCRMGEntities())
	{
		List<PaginaSmartDato> resultado = new List<PaginaSmartDato>();

		IQueryable<VPART> paginasQuery = context.VPARTs;

		var soloPaginasQuery = paginasQuery.Where(c => c.Tipo ==
		"Pagina");

		foreach (VPART pagina in soloPaginasQuery)
		{
			resultado.Add(new PaginaSmartDato { Codigo =
			pagina.Codigo, Titulo = pagina.Descripcion });
		}

		return resultado;
	}
```

La lista es serializada automáticamente y se envía al cliente en formato JSON.

6.2.3.2.2 Petición de una Pagina Smart

En este caso recibimos un código de página en la petición que lo procesará el controlador PaginaSmartController. Con este código accederemos hasta la capa de negocio para obtener la configuración en formato XML de la base de datos. La parte más importante de este proceso es la transformación de ese XML a objeto en C# que explicaré a continuación.

[1] Language Integrated Query. LINQ extiende el lenguaje a través de las llamadas expresiones de consulta, que son parecidas a las sentencias SQL y pueden ser usadas para extraer y procesar convenientemente datos de arrays, clases enumerables, documentos XML, bases de datos relacionales y fuentes de terceros.

Tratamiento XML a objeto

Esta es la explicación justo antes de que el objeto XML se convierta en una `PaginaSmart`. Desde la capa de datos, recuperamos el XML y luego es en la capa de negocio donde lo procesamos con la siguiente función:

```
/**
* Obtiene la pagina Smart dado el codigo de la pagina
*/
public PaginaSmart GetPaginaSmart(string codigoPagina)...
```

El `string` resultante de la capa de datos lo cargamos en el objeto de tipo `XDocument`:

```
string xml_string = datos.GetPaginaSmartWeb(codigoPagina);
TextReader sr;
sr = new StringReader(xml_string);
XDocument xml = XDocument.Load(sr);
sr.Close();
```

Y ahora, esto nos permite ir rellenando el objeto `PaginaSmart` con los datos de este XML:

```
XElement xml_tabItem = xml.Element("cfgTabItem");

resultado.CodigoPagina = xml_tabItem.Attribute("CodigoPagina").Value;
resultado.Titulo = xml_tabItem.Attribute("Titulo").Value;
resultado.Ratio = xml_tabItem.Attribute("Ratio").Value;

foreach (XElement colum in xml.Descendants("cfgColumnDefinition"))
{
      DefinicionColumna columDef = new DefinicionColumna();
      columDef.Ancho =
      colum.Element("Definition").Attribute("Ancho").Value;
      resultado.Columnas.Add(columDef);
}

foreach (XElement row in xml.Descendants("cfgRowDefinition"))
{
      DefinicionFila rowDef = new DefinicionFila();
      rowDef.Alto = row.Element("Definition").Attribute("Alto").Value;
      resultado.Filas.Add(rowDef);
```

Ahora nos recorreremos las entidades de la configuración para convertirlas en objetos `WPart`:

```
foreach (XElement entity in xml.Descendants("cfgMarcoControlMenu"))
{
      WPart wpart = new WPart();
       wpart = GetWPartSmartWeb(entity);
       resultado.Entidades.Add(wpart);
}
```

La función de `GetWPartSmartWeb`(), dado una entidad (un `WPart`) instanciada en blanco, llamará a la función correspondiente, para que esta se encargue de hacer todas las asignaciones

correspondientes dependiendo del tipo. Todo esto estará comentado en el código que se entregue con mucho más detalle. Esto es simplemente una idea de cómo funciona el sistema.

6.2.3.2.3 Petición de datos de un grafico

Como se ve en el diagrama del apartado de diseño, el servicio hace uso de una DLL proporcionada por la empresa, llamada `VectorCBuss.dll` que nos devuelve un `string` con los datos que necesitamos. La clase `WPartGraficoController` del servicio REST, se encarga de transformar ese resultado en un tipo `Dictionary<string,List<WPartGraficoDato>>.`

```
public Dictionary<string, List<WPartGraficoDato>> Post(WPartGrafico wPart)
{
	...

	Dictionary<string, List<WPartGraficoDato>> resultado = new
	Dictionary<string, List<WPartGraficoDato>>();

	for (int i = 0; i < listaSeriesVisibles.Count; i++)
	{
		...
		for (int j = 0; j < valoresEjeX.Count(); j++)
		{
			...
			Categoria = valoresEjeX[j],
			Valor = datos[i][j]
			...
		}

		...
	}

	return resultado;
}
```

Luego es serializado y enviado mediante el método Post al cliente.

En estas funciones hemos podido ver que se usa un tipo de objeto llamado `WPartGraficoDato`, el cual no hemos definido hasta ahora, pero es el siguiente:

```
public class WPartGraficoDato
{
	public string Categoria { get; set; }
	public decimal Valor { get; set; }
}
```

Simplemente se encarga de la categoría y el valor de los gráficos, la razón de que exista es para “tipar” los datos devueltos por el servicio REST y los pueda reconocer el dispositivo móvil para realizar el “binding” de datos.

6.2.3.2.4 Petición de datos de una consulta

El servicio recibe un objeto del tipo `WPartConsultaConFiltro` donde la capa de presentación, se encarga de separar por un lado la entidad y por otro el filtro:

```
[HttpPost]
public List<Dictionary<string, string>> Post(WPartConsultaConFiltro
wPart_filtro)
{
      bcFunc negocio = new bcFunc();

      string filtro = wPart_filtro.Filtro;

      if(filtro == "")
            return
      negocio.ObtenerDatosParaWPartConsulta(wPart_filtro.WPart);
      else
            return
```

Y en función del filtro, esto es si es una entidad dependiente o no, llamamos a la función correspondiente de la capa de negocio.

Como se ve en el diagrama que se muestra en el punto de diseño, el servicio hace uso de una DLL proporcionada por la empresa, en este caso la clase llamada `VectorCBuss.dll` que nos devuelve un `string` con los datos que necesitamos. La clase `WPartConsultaController` del servicio REST, se encarga de transformar ese resultado en un tipo `List<Dictionary<string, string>>.`

Luego es serializado y enviado mediante el método Post al cliente.

6.2.3.2.5 Petición de datos de una consulta ficha

El servicio recibe un objeto del tipo `WPartConsultaFichaConFiltro` donde la capa de presentación, se encarga de separar por un lado la entidad y por otro el filtro:

```
[HttpPost]
public Dictionary<string, string> Post(WPartConsultaFichaConFiltro
wPart_filtro)
{
      bcFunc negocio = new bcFunc();

      string filtro = wPart_filtro.Filtro;

      if(filtro == "")
            return
      negocio.ObtenerDatosParaWPartConsultaFicha(wPart_filtro.WPart);
      else
            return
```

Y en función del filtro, esto es si es una entidad dependiente o no, llamamos a la función correspondiente de la capa de negocio.

Como se ve en el diagrama que se muestra en el punto de diseño, el servicio hace uso de una DLL proporcionada por la empresa, en este caso la clase llamada `VectorCBuss.dll` que nos devuelve un `string` con los datos que necesitamos. La clase `WPartConsultaFichaController` del servicio REST, se encarga de transformar ese resultado en un tipo `Dictionary<string, string>.`

Luego es serializado y enviado mediante el método Post al cliente.

6.2.4 Pruebas

El primero de los niveles del servicio REST, se compone de 2 proyectos. Uno llamado "WebPruebas", donde se nos permite ver en el navegador del ordenador los resultados que obtenemos desde el servicio, y en el cual, escogemos diferentes formatos de visualización de los datos a nuestra conveniencia. Y otro, llamado "eSmartCoreService", que es en el que irán todas las implementaciones y modificaciones que hayamos probado con éxito en el proyecto de "WebPruebas", y al que la aplicación móvil accederá para obtener los datos en un formato específico.

6.2.4.1 Pruebas unitarias

El objetivo principal de estas pruebas es validar el comportamiento de las funciones que componen cada módulo.

Para realizar estas pruebas hemos diseñado unos escenarios en los que se pueda verificar que el servicio cumple con los requisitos exigidos por la empresa.

6.2.4.2 Diseño de los casos de prueba

Los casos de prueba aquí mencionados son los relacionados con las pruebas de integración del sistema. Los casos de prueba en que se han dividido el plan de pruebas pueden agruparse en dos tipos:

- Pruebas de ejecución correctas
- Pruebas con error

Las pruebas de ejecuciones correctas tienen como objetivo verificar las funcionalidades del sistema, es decir que, dado un caso, cumpla el comportamiento esperado.

Las pruebas de ejecuciones con error tienen como objetivo verificar el correcto tratamiento de los errores por parte del servidor, es decir, dado un parámetro de entrada mediante la URL, la salida generada (el JSON) debe coincidir con las especificaciones funcionales.

Pruebas de ejecución correctas

La aplicación tiene unos casos de uso muy concretos, por lo que nos hemos asegurado de que cada uno funcione correctamente.

- Solicitar lista de páginas: Funciona correctamente.
- Solicitar una página: Funciona Correctamente.
- Solicitar datos de un gráfico: Funciona Correctamente.
- Solicitar datos de una consulta: Funciona Correctamente.
- Solicitar datos de una consulta con ficha: Funciona Correctamente.

Pruebas de ejecución con error

Estas pruebas han sido planteadas para verificar el comportamiento del servicio REST en situaciones en las que no se recibe lo esperado de la aplicación cliente, o reciba un ataque. En los casos de error interno del servidor, muestra una pantalla como la siguiente:

Imagen 6.2.1. Error Servidor.

6.2.4.3 Pruebas de validación

Una vez realizadas las pruebas de integración se han realizado las pruebas de validación. El objetivo principal de estas pruebas es comprobar que se cumplen los requisitos funcionales y el rendimiento que se ha definido en un principio con la empresa. Para validar el sistema se ha utilizado la técnica de la caja negra utilizando como criterio de validación la aceptación del sistema por parte de la empresa.

6.3 DESARROLLO DEL CLIENTE

Una vez desarrollado el servicio REST, hemos decidido desarrollar una aplicación móvil que sea capaz de consumir algunos de los servicios que ofrece, ya que el objetivo de la empresa es llevar su producto a plataformas móviles.

6.3.1 Análisis

En este caso, no es estrictamente necesario la realización de una aplicación móvil para la empresa, pero sí que es necesario para el desarrollo de mi proyecto. Por lo que los requisitos funcionales de esta aplicación puedo definirlos en parte a mi gusto. Y ya que la empresa había decidido implementar unos casos de uso concretos, los requisitos que yo establezco a mi aplicación móvil serán para probar que cada uno de los requisitos que se han implementado en el servicio funciona correctamente. Los requisitos serán los siguientes:

- Pedir una lista de todas las paginas disponibles en la base de datos
- Seleccionar una página Smart
- Seleccionar una entidad y poder visualizarla correctamente
    - Un gráfico
    - Una consulta de tipo lista
    - Una consulta de tipo ficha
- Ver los detalles de una línea que no entre en la pantalla

Esta aplicación obligatoriamente necesitará conexión a internet, y no accederá a ningún archivo del móvil, tampoco recopilará ningún tipo de datos del usuario, por lo que el único permiso que pedirá al usuario será el de conexión a internet.

No será necesario implementar métodos adicionales de seguridad ya que la aplicación de momento se va a crear con fines demostrativos y solo será instalada en terminales de confianza, no será distribuida al público.

6.3.2 Diseño

En este capítulo se muestra el diseño de la aplicación, estudiando en profundidad cada uno de los casos de uso resultantes tras la captura de requisitos. Como ya hemos visto anteriormente en el esquema del servicio REST.

6.3.2.1 Funcionamiento de la aplicación

En este apartado se describen las funcionalidades que tiene la aplicación eSmartCore que hemos desarrollado. Explicaremos cuales se han implementado para la demo, y cuales se han dejado sin implementar.

Como solo hay un usuario, no será necesario dividir los casos de uso, puesto que todos los realizará el mismo.

Usuario de pruebas

Al abrir la aplicación, puede pedir una lista de las paginas disponibles, estas se le mostraran por título. Cada elemento de la lista enlaza con su respectiva página Smart.

Desde esta pantalla, puede navegar hacia una página en concreto, donde dependiendo del tipo de página que sea, ocurrirán diferentes cosas. Nosotros hemos implementado tres de los múltiples tipos de entidades que hay, que son las vistas de los gráficos, consultas y consultas ficha.

- Si se selecciona una página de gráficos, se mostrará una lista con las diferentes opciones que puede visualizar.
- Si se selecciona una página de consultas, se mostrará una lista con las diferentes consultas disponibles. Puede ocurrir que tengan otro objeto relacionado, como la ficha de la consulta.
- Si se selecciona uno de los gráficos de la primera opción, se mostrará a pantalla completa un gráfico interactivo en el que se podrá ver la representación gráfica de los datos.
- Si se selecciona una de las consultas, dependiendo de si tiene fichas asociadas o no, se mostrará la consulta en sí, o bien, una lista de fichas de consulta para escoger cual se desea ver.

La ficha se podrá consultar haciendo clic en el título de la ficha desde la anterior página.

6.3.2.2 Diagrama de flujo

En el siguiente diagrama, intentamos explicar de una manera más clara el funcionamiento de la aplicación:

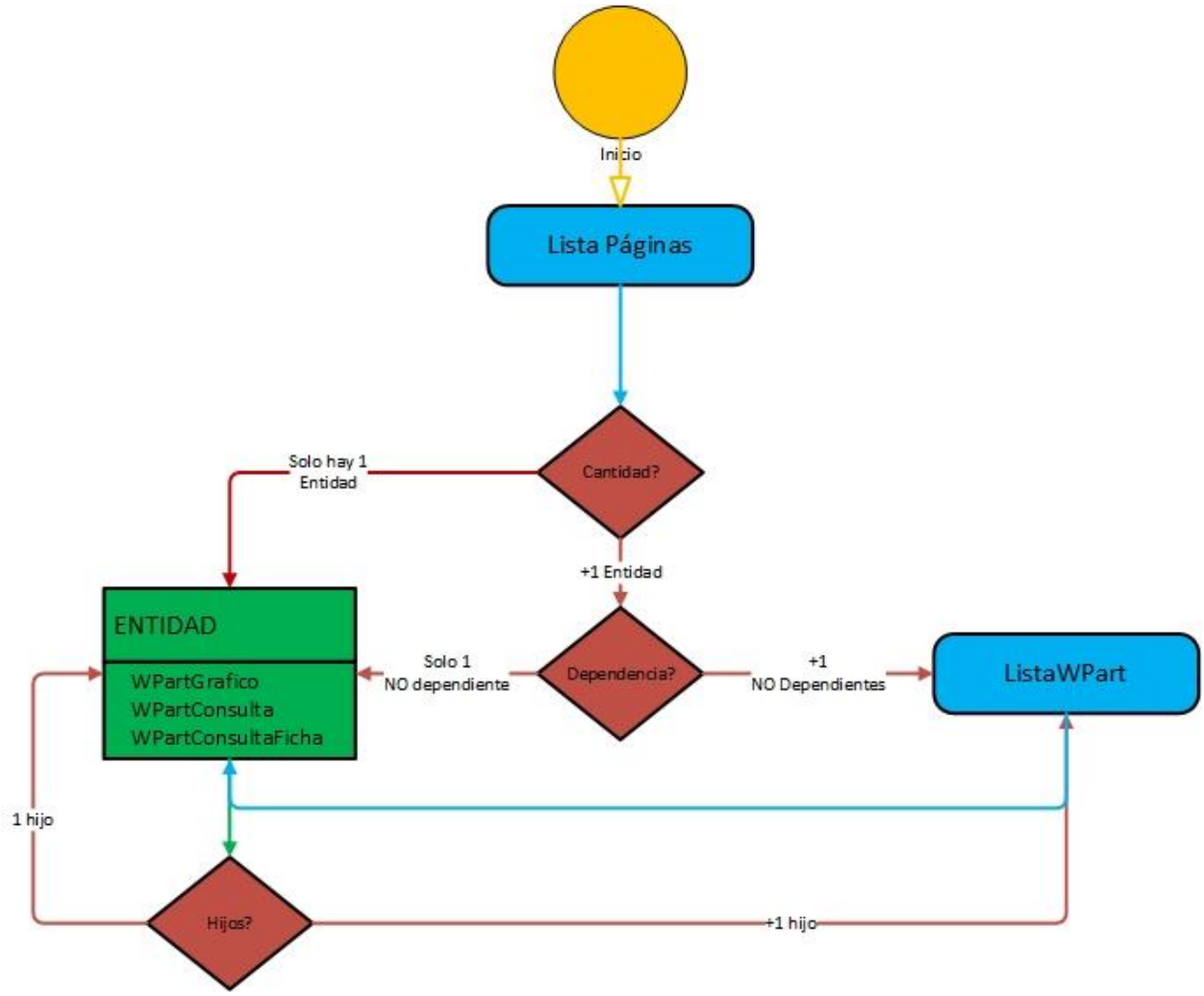


Diagrama 6.3. Diagrama de flujo.

Las opciones serían, desde la lista de páginas, al seleccionar una y, dependiendo de la cantidad de entidades que haya, se procederá a mostrar la entidad si solo hay una, o preguntaremos si de todas las que hay, hay más de una que no tenga dependencias. Si no hay más de una, se mostrará la única entidad que no tenga dependencias, si no, se mostrará una lista de WPart, en la que podremos escoger la WPart a la que queremos navegar.

Una vez en la entidad, si tiene opción de seleccionar un registro, dependiendo de si tiene uno o más hijos, veremos directamente la entidad, o nos llevará a una nueva lista en la que podremos escoger qué entidad queremos ver.

Destacar que esto último, solo será posible en el caso de que el tipo de WPart sea de tipo WPartConsulta. Los gráficos y las fichas son un estado final.

6.3.2.3 Estudio de los casos de uso

En este apartado se hace un estudio exhaustivo de cada uno de los casos de uso. Para cada caso de uso se ha realizado:

- Una breve descripción del caso de uso
- Diagrama del actor que interviene en cada caso
- Diseño de la interfaz de usuario
- Flujo de eventos
- Diagrama de secuencia

Casos de uso del usuario de pruebas

6.3.2.3.1 Caso de Uso: Entrada a la aplicación

Proceso mediante el que el usuario entra a la aplicación “eSmartCore”.

Imagen 6.3.2.1. Casos de uso aplicación móvil.

Diseño de la interfaz:

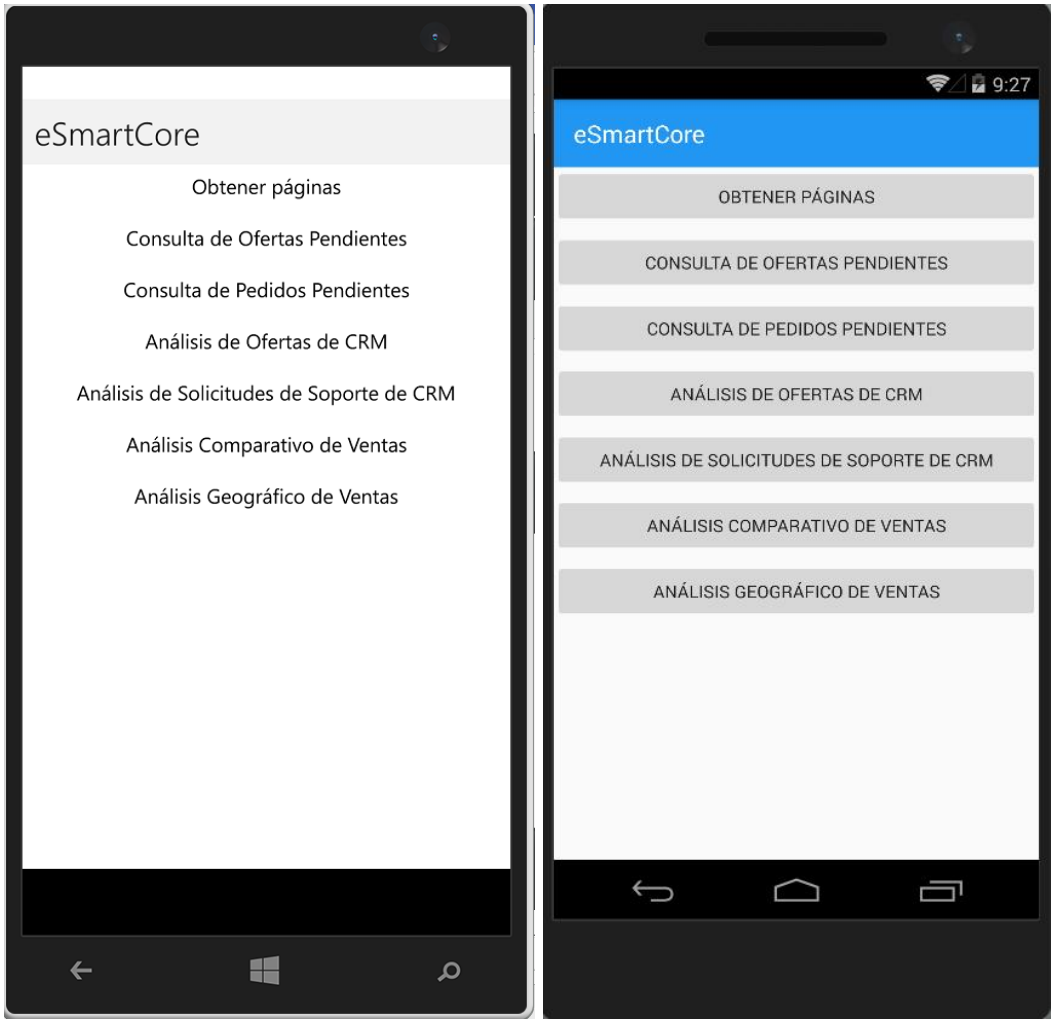


Imagen 6.3.2.2. Derecha Windows 10, Izquierda Android 4.4

Flujo de eventos:

1. El usuario desde el móvil, pulsa en la aplicación para entrar, si el servicio REST no estuviese disponible, saldría un error y habría que intentar reconectar.
2. Tras pulsar en el botón de obtener páginas, si todo es correcto procederá a mostrar la lista de páginas disponibles para el usuario.

Diagrama de secuencia:

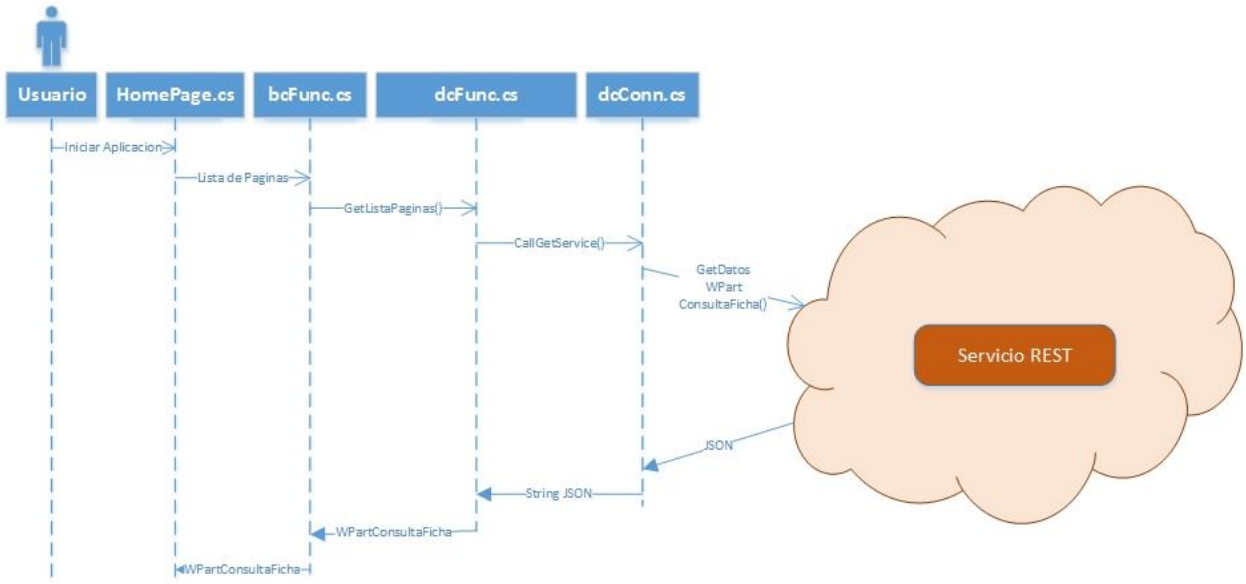


Imagen 6.3.2.3. Ver imagen en tamaño completo, pág. 148.

6.3.2.3.2 Caso de Uso: Seleccionar una página

Este es un proceso en el cual puede haber varios resultados diferentes, por un lado, está el caso en el que la pagina solo conste de una entidad, o en el caso de que haya más de una que solo una sea NO dependiente, y, por otro lado, tenemos el caso en el que la pagina conste de varias entidades y más de una sean NO dependientes. Ambos casos los vamos a explicar a continuación:

Primer caso

En este caso, seleccionamos una página, en la que solo hay una entidad, o en la que hay más de una entidad, pero solo hay una que no sea dependiente, esto es, que no necesite recibir datos de ninguna otra entidad. Lo que sucedería, sería que se nos mostraría por pantalla la única entidad que haya, ya sea un gráfico, una consulta o una consulta ficha.

Imagen 6.3.2.4. Caso de uso

Diseño de la interfaz:

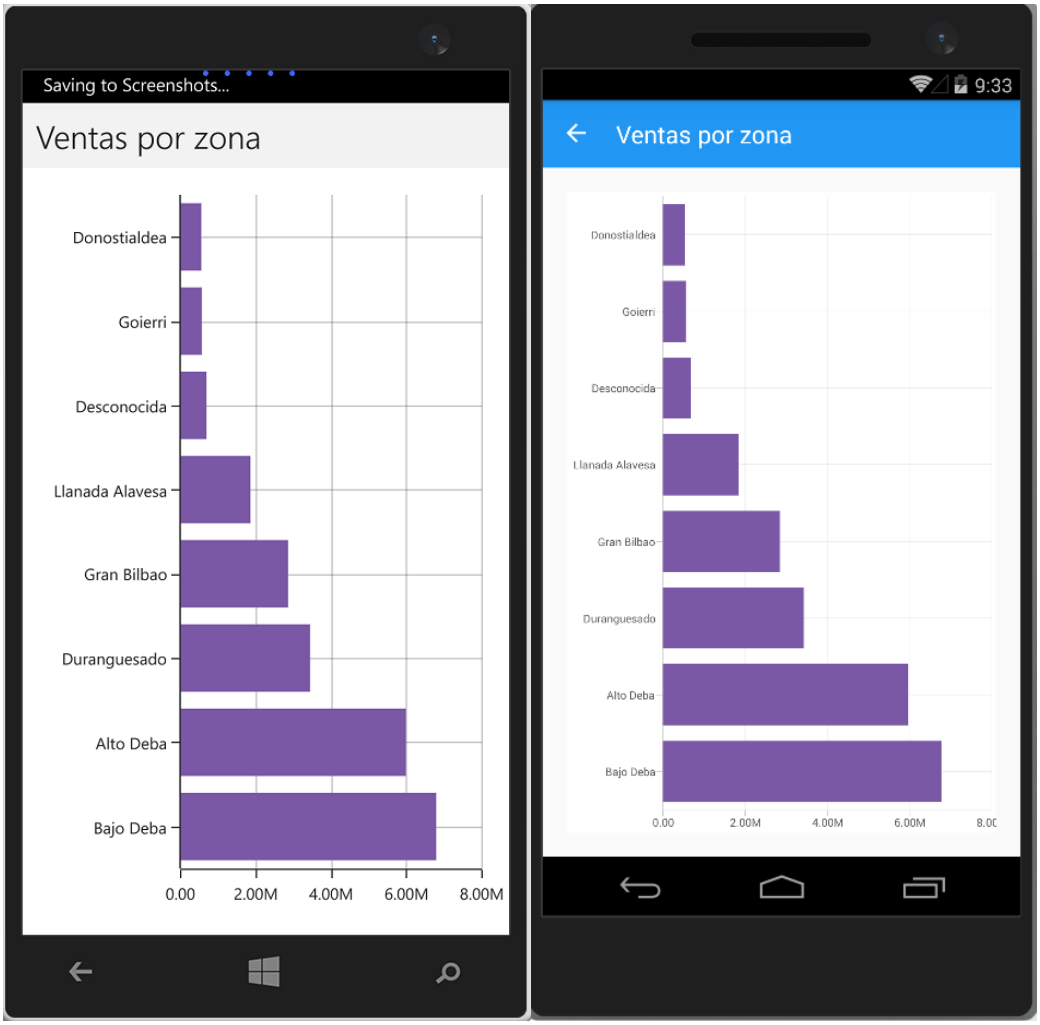


Imagen 6.3.2.5. Derecha Windows 10, Izquierda Android 4.4

Flujo de eventos:

1. El usuario desde la página principal pulsa en una página que desea ver
2. Se mostrará la entidad que contiene esa página.

Diagrama de secuencia:

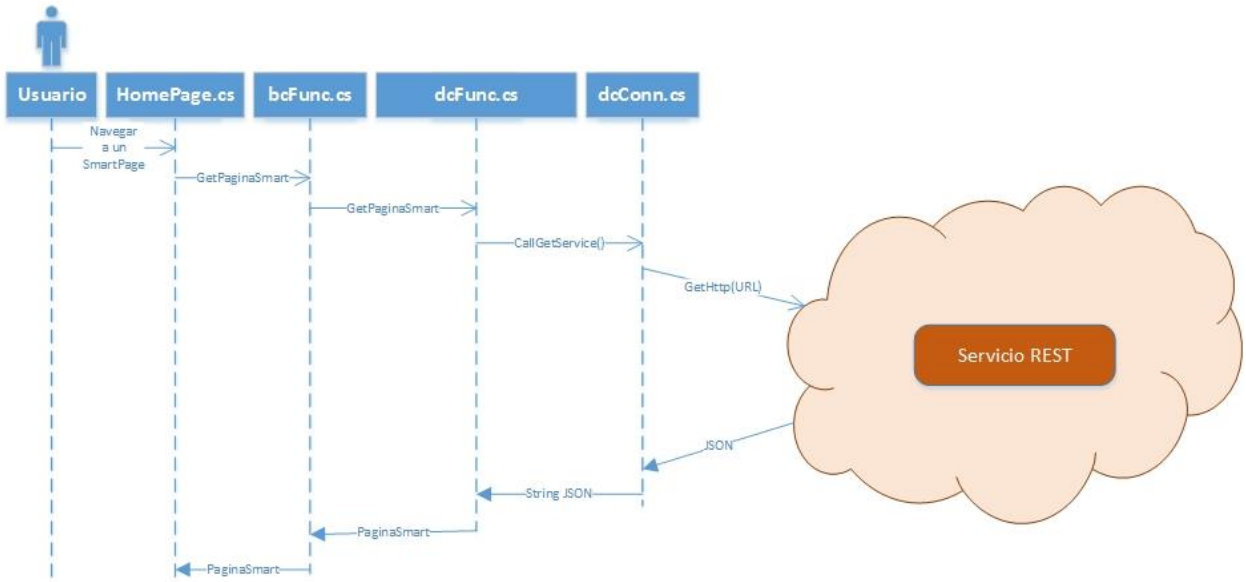


Imagen 6.3.2.6. Ver imagen en tamaño completo, pág. 149.

Segundo caso

En este caso, seleccionamos una página que tiene más de una entidad NO dependiente, esto quiere decir que ninguna otra entidad tiene que proporcionarle datos. Y veremos por pantalla una lista con las diferentes entidades de las que está compuesta la página.

Imagen 6.3.2.7. Caso de uso

Diseño de la interfaz:

Imagen 6.3.2.8. Derecha Android 4.4, Izquierda Windows 10

Flujo de eventos:

1. El usuario desde la página principal pulsa en una página que desea ver
2. Se mostrará la lista de entidades que contiene esa página

Diagrama de secuencia:

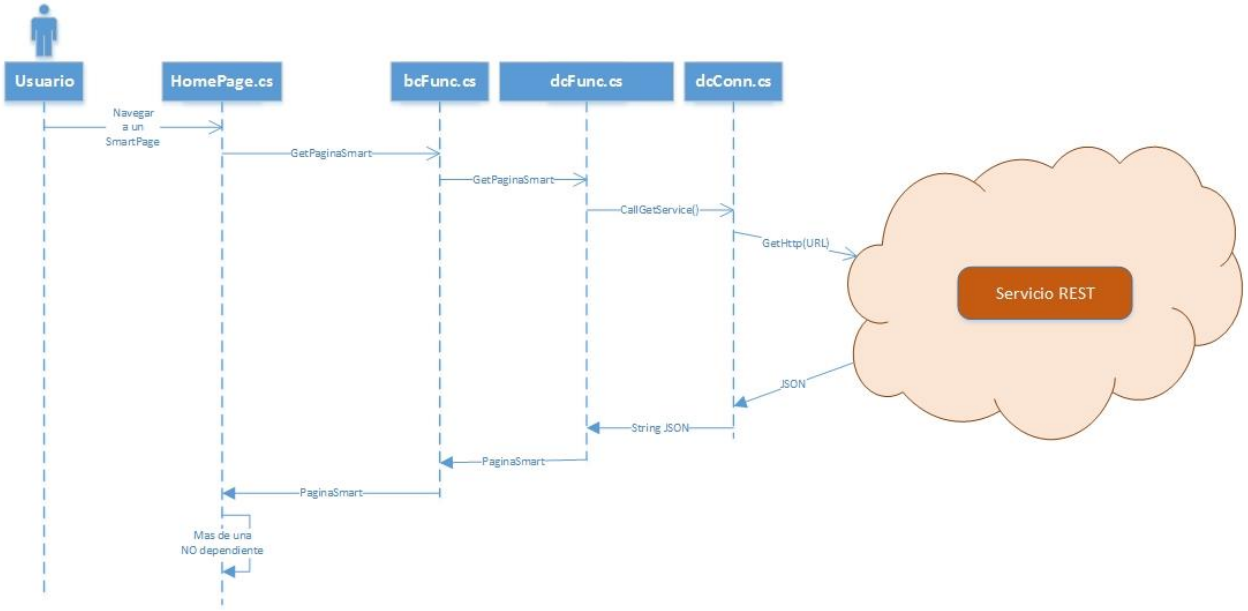


Imagen 6.3.2.9. Ver imagen en tamaño completo, pág. 150.

6.3.2.3.3 Caso de Uso: Seleccionar un Grafico

En este caso, es el grafico único, una vez seleccionas una de las páginas de la lista principal, o de otra lista de entidades, se ha seleccionado una que solo contiene un WPart y es de tipo gráfico.

Imagen 6.3.2.10. Caso de uso

Diseño de la interfaz:

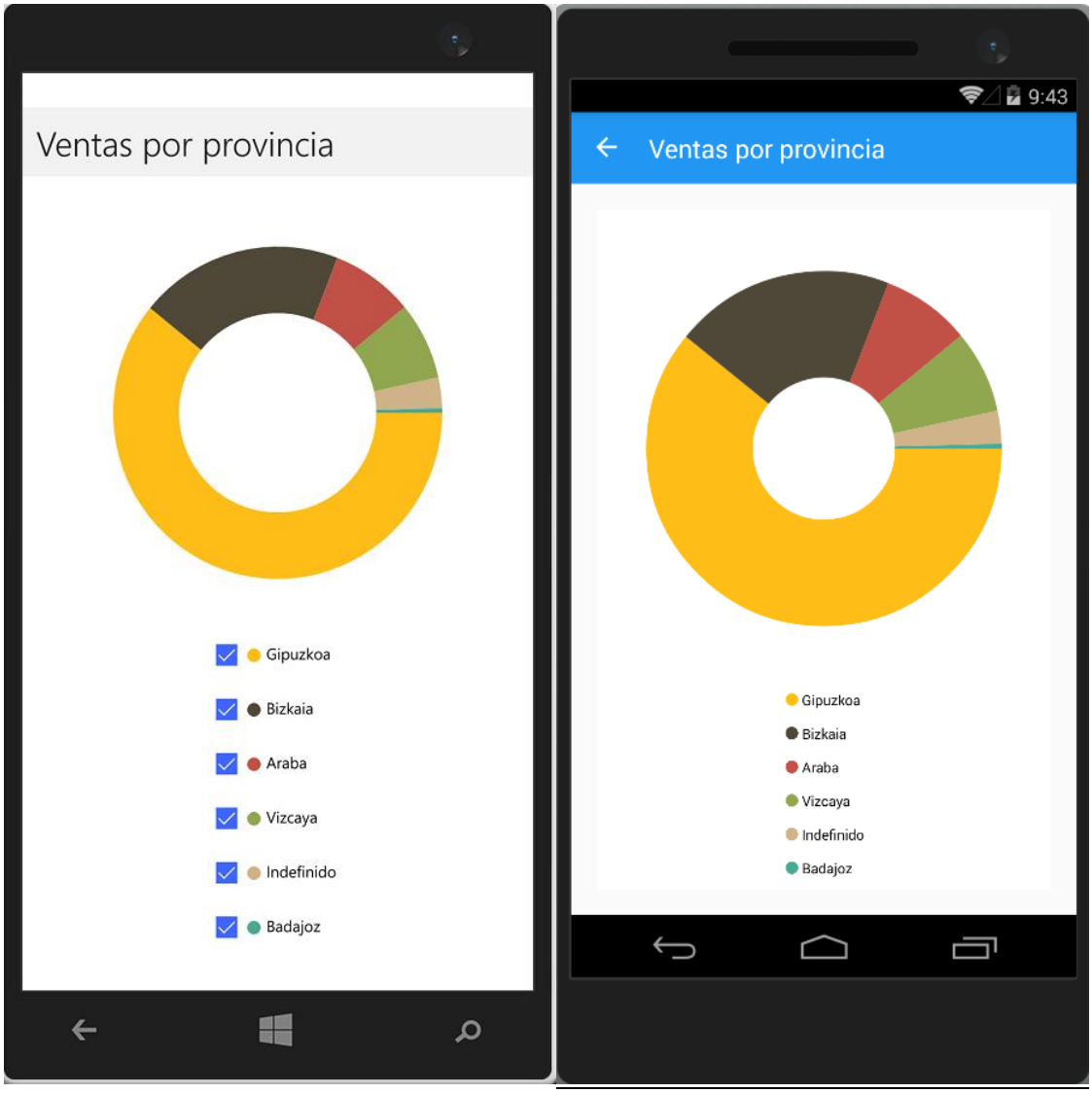


Imagen 6.3.2.11. Derecha Windows 10, Izquierda Android 4.4

Flujo de eventos:

1. El usuario desde la página principal, o desde una lista de entidades, pulsa en la que desea ver
2. Se mostrará el grafico correspondiente con la entidad

Diagrama de secuencia:

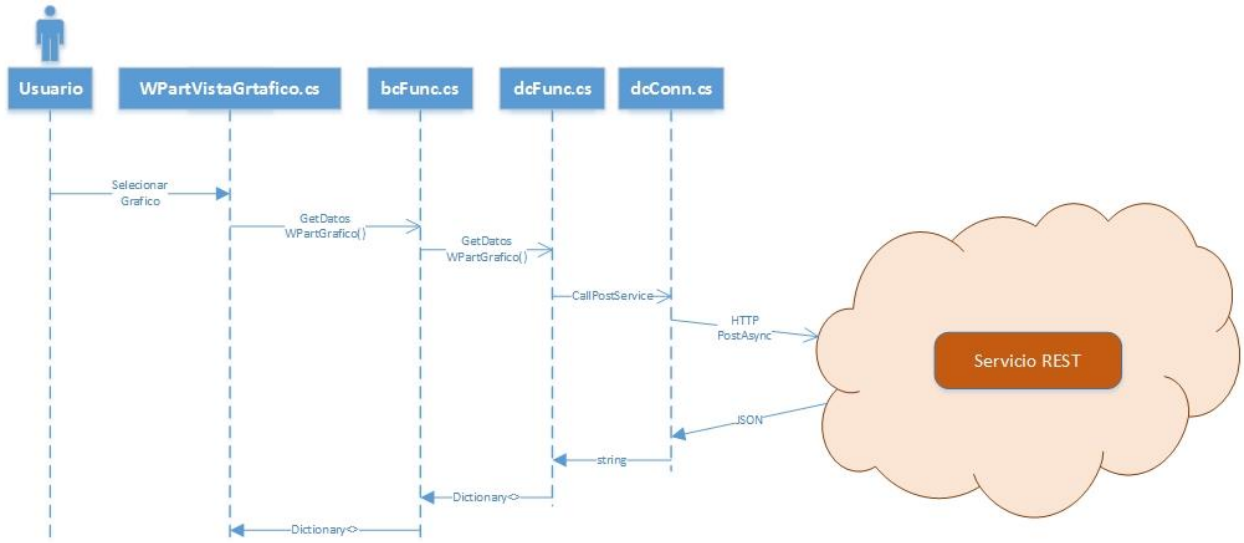


Imagen 6.3.2.12. Ver imagen en tamaño completo, pág. 151.

6.3.2.3.4 Seleccionar una Consulta

En este caso, es una consulta, una vez seleccionas una de las páginas de la lista principal, o de otra lista de entidades, se ha seleccionado una que solo contiene un WPart y es de tipo consulta.

Imagen 6.3.2.13. Caso de uso

Diseño de la interfaz:

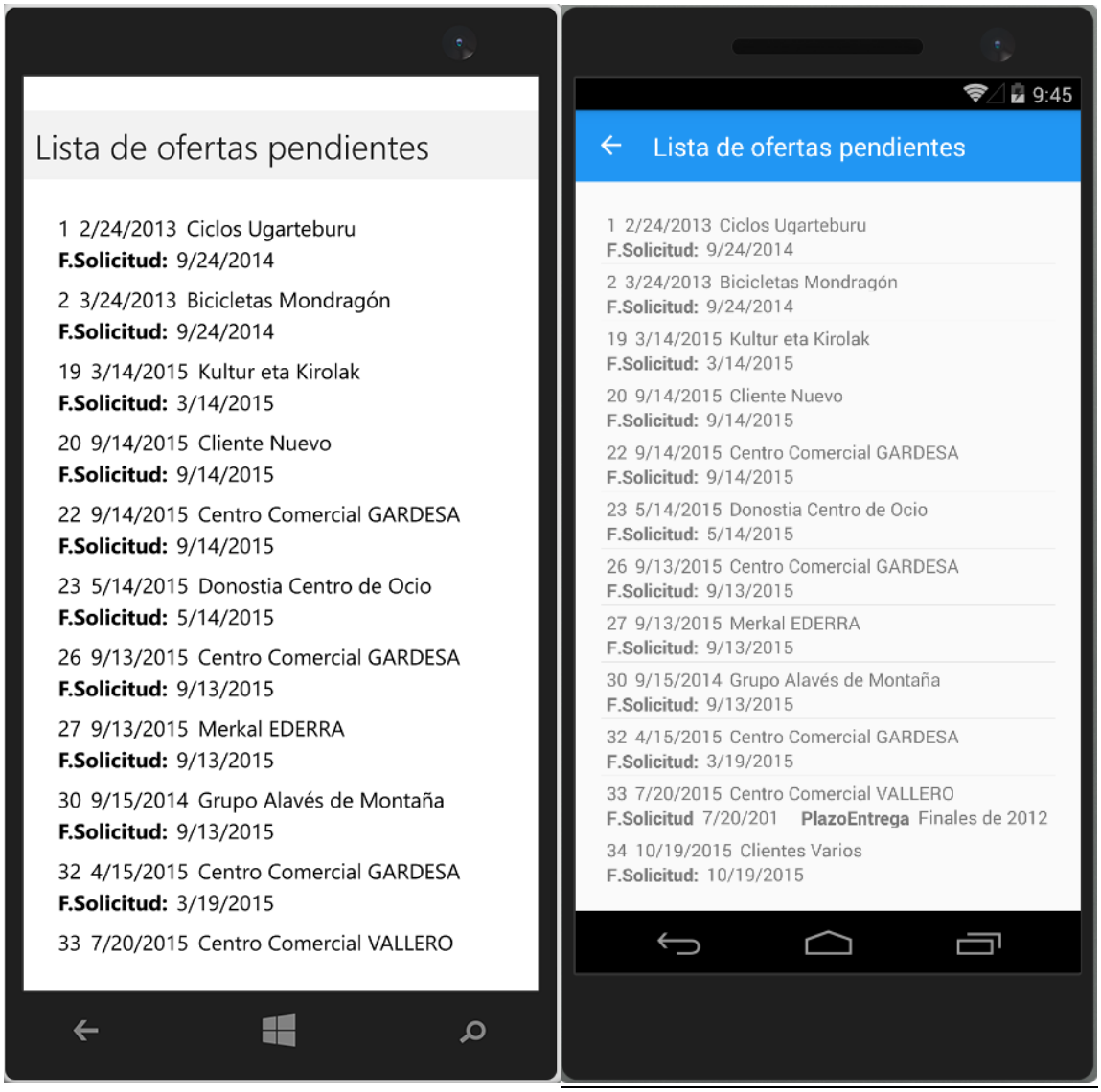


Imagen 6.3.2.14. Derecha Android 4.4, Izquierda Windows 10

Flujo de eventos:

1. El usuario desde la página principal, o desde una lista de entidades, pulsa en la que desea ver
2. Se mostrará la consulta correspondiente con la entidad

Diagrama de secuencia:

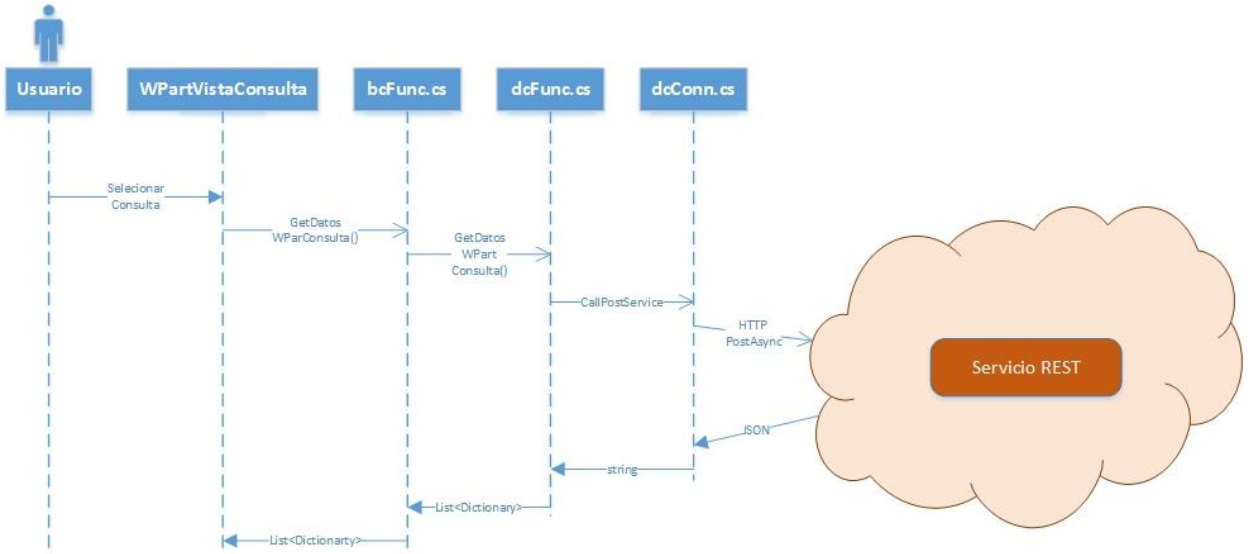


Imagen 6.3.2.15. Ver imagen en tamaño completo, pág. 152.

6.3.2.3.5 Caso de Uso: Seleccionar una Consulta Ficha

En este caso, es una consulta ficha, una vez seleccionas una de las páginas de la lista principal, o de otra lista de entidades, se ha seleccionado una que solo contiene un WPart y es de tipo consulta ficha.

Imagen 6.3.2.16. Caso de uso

Diseño de la interfaz:

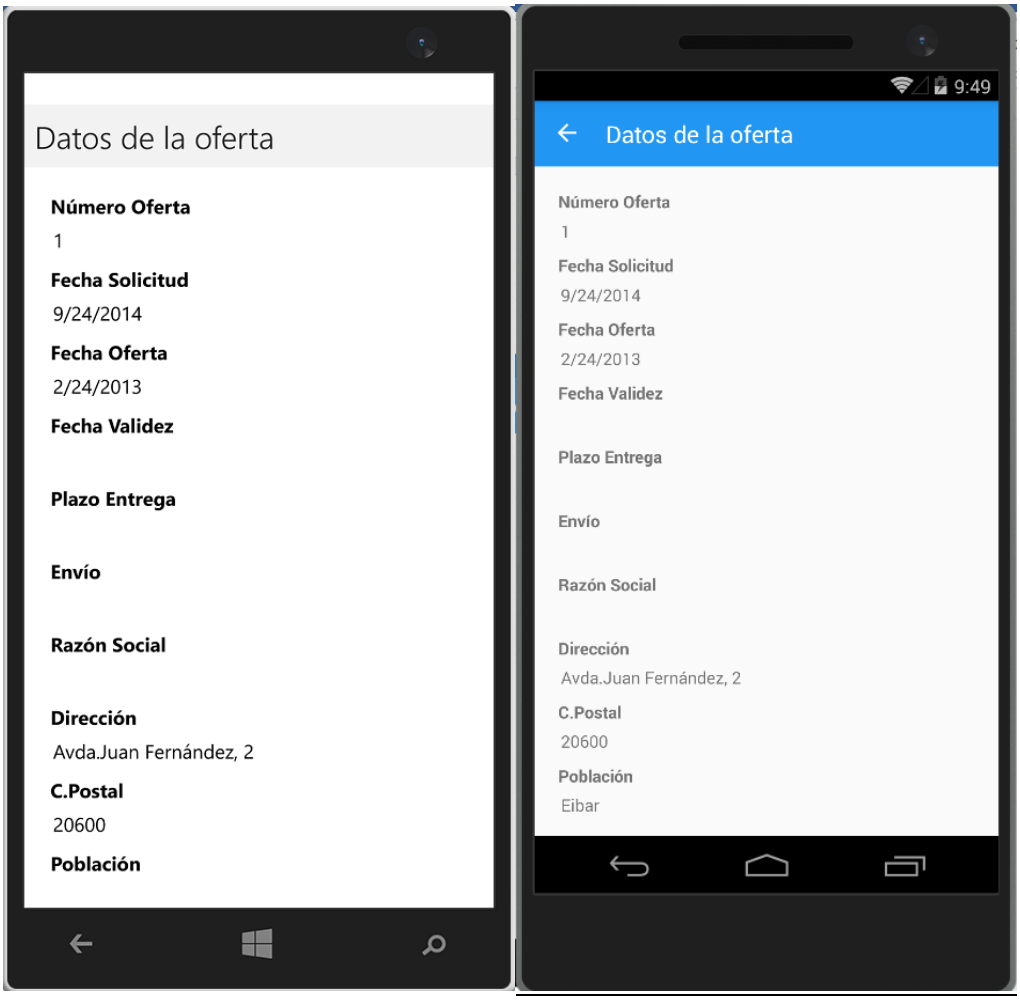


Imagen 6.3.2.17. Derecha Android 4.4, Izquierda Windows 10

Flujo de eventos:

1. El usuario desde la página principal, o desde una lista de entidades, pulsa en la que desea ver
2. Se mostrará la consulta ficha correspondiente a la consulta seleccionada

Diagrama de secuencia:

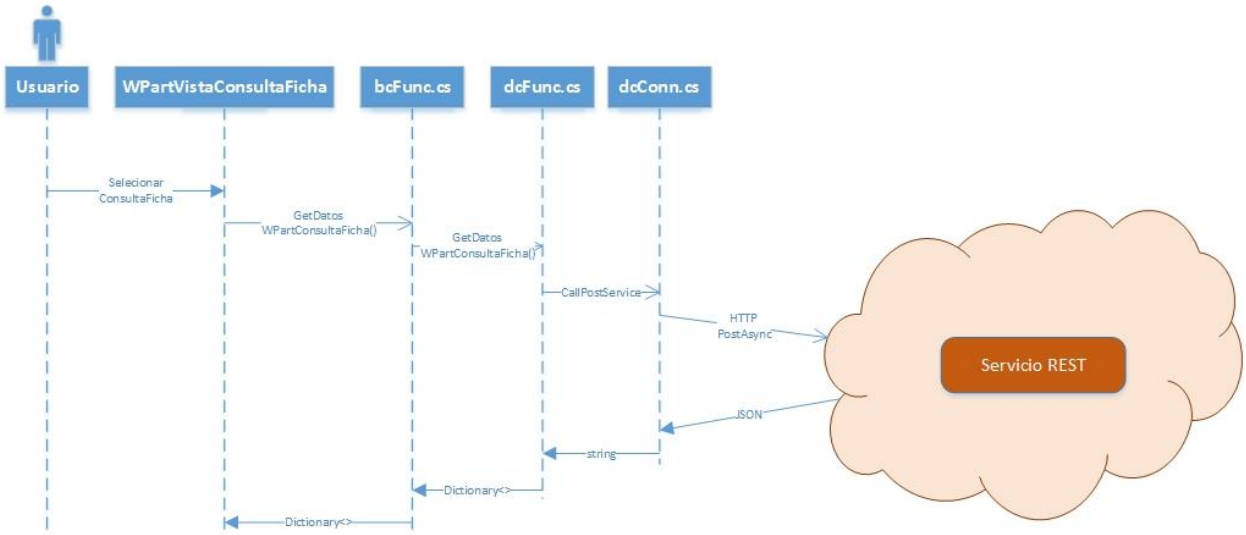


Imagen 6.3.2.18. Ver imagen en tamaño completo, pág. 153.

6.3.2.3.6 El caso de que una consulta tenga hijos

Dentro de las consultas, pueden tener varios hijos, que son otras entidades que dependen de ellas. Si solo tiene un hijo, al seleccionar un registro en la consulta navegaremos directamente al hijo, pero en el caso de que tenga más de uno, se irá a una página intermedia con una lista de entidades dependientes para que seleccione cual quiere visualizar y después navegar a ella.

Las páginas que dependen de otra página, reciben un filtro.

6.3.3 Implementación

En este capítulo, veremos cómo se han implementado los casos de uso que se han explicado en el apartado de diseño, también veremos que componentes hemos utilizado a la hora de programar la interfaz.

6.3.3.1 Controles

En la aplicación móvil, los componentes que podremos encontrar son los siguientes:

Páginas

- ContentPage: Son las páginas de contenido.
- NavigationPage: Son páginas a las que se navega desde otra página. Las ventajas que tiene es que no es necesario recordar de que página vienes para volver hacia atrás, es automático.

Layouts

- ContentView: No es un componente destinado a algo en concreto, sino a hacer de base para otros.
- ScrollView: Es un contendor donde se pueden insertar varias vistas, el cual, si no entra todo en la pantalla, permite deslizar hacia abajo con el dedo y así poder seguir viendo líneas.
- StackLayout: Es un contenedor donde todas las vistas que se introduzcan se apilan una debajo de la otra, sin importar el tamaño.

Vistas

- Button: Es el clásico botón, que nos permite lanzar un evento al seleccionarlo.
- Label: Una etiqueta donde mostrar el texto que queramos.
- ListView: Es un visor de elementos, el cual los despliega en forma de una lista vertical.

Vista de los gráficos:

Syncfusion es un componente que se añade a Visual Studio y que permite usar sus herramientas para el desarrollo. En este caso, nosotros hacemos uso de los componentes que ofrece para Xamarin.Forms. Syncfusion no es totalmente gratuito, pero ofrece una licencia para estudiantes que quieran desarrollar algún proyecto.

Entre los controles que usa nuestra aplicación, los que pertenecen al conjunto de controles de Syncfusion es el SfChart. En un principio también hacíamos uso del componente SfListView, pero fue sustituido por el control original de Xamarin, ListView.

SfChart es un componente que nos permite representar gráficos de una manera más sencilla y con una estética muy agradable. También son altamente personalizables. Todos los gráficos que se ven en la aplicación son de esta clase.

6.3.3.2 Entrada en la aplicación

La entrada en la aplicación consiste en crear la página de inicio, desde la cual derivarán el resto:

```
public HomePage()
        {
            ...

            Button boton = new Button();
            boton.Text = "Obtener páginas";
            boton.Clicked += (sender, e) =>
            {
                IniciarListaPaginas();
            };

            marco.Children.Add(boton);

            ScrollView sv = new ScrollView();
            Contenido = new StackLayout();
            Contenido.Orientation = StackOrientation.Vertical;

            ...

            this.Content = marco;

        }
```

la llamada a la función `IniciarListaPaginas`() es la encargada de proporcionar la lista de páginas.

```
private void IniciarListaPaginas()
        {
            ...

            List<PaginaSmartDato> listaPaginas =
Negocio.GetListaPaginas().Result;

            for (int i = 0; i < listaPaginas.Count; i++)
            {
                TagButton boton = new TagButton();
                boton.Text = listaPaginas[i].Titulo;
                boton.Tag = listaPaginas[i].Codigo;
                boton.Clicked += async (sender, e) =>
                {
                    string codigo = ((TagButton)sender).Tag;

                    await Navigation.PushAsync(new SmartPage(codigo));
                };

                Contenido.Children.Add(boton);
            }
        }
```

Estamos creando una lista de botones al vuelo, en los que se podrá seleccionar y en los que se ejecutará un “PushAsync”, que es la forma que tiene Xamarin de llamar a una nueva página, en este caso una `SmartPage` en la que se le proporciona el código de la página seleccionada.

Dentro de esta primera hacemos una llamada a `GetListaPaginas`(), que es la función que podemos ver en el diagrama de secuencia del caso de uso "Entrada en la aplicación". Esta función se encarga de llamar a la capa de negocio y nos devolverá una lista de `PaginaSmartDato`:

```
public async Task<List<PaginaSmartDato>> GetListaPaginas()
{
	List<PaginaSmartDato> resultado = await Datos.GetListaPaginas();

	return resultado;
}
```

En la capa de negocio, nos encargaremos de hacer la llamada a la capa de datos:

```
public async Task<List<PaginaSmartDato>> GetListaPaginas()
{
	List<PaginaSmartDato> resultado = new List<PaginaSmartDato>();

	string l = await Conexion.CallGetService("paginasmart");

	resultado = JsonConvert.DeserializeObject<List<PaginaSmartDato>>(l);

	return resultado;
}
```

Aquí se observa cómo se crea la lista de datos, y como hacemos la llamada a la función `CallGetService`() con el dato "paginasmart" lo cual nos proporcionara la lista en formato JSON para que nosotros la deserialicemos. En este caso, la serialización y la de-serialización son completamente automáticas, no ha sido necesaria ninguna modificación, ni redefinición, de ningún método de la clase `JsonConvert`.

En el método `CallGetService`(), que podemos observar aquí:

```
public async Task<string> CallGetService(string urlApi)
{
	using (var client = new HttpClient())
	{
		string url = "";
	#if __ANDROID__
		url = DominioApiAndroid + urlApi;
	#else
		url = DominioApiWindows + urlApi;
	#endif
		var result = Task.Run(() => client.GetAsync(url)).Result;
		return await result.Content.ReadAsStringAsync();
	}
}
```

En este trozo de código, he utilizado uno que no es de la versión final, para que se vea cómo hacemos uso de las ventajas de Xamarin. Podemos ver que tenemos unas instrucciones `if`, `else` y `endif`, en las cuales se nos permite definir diferentes URL para la misma aplicación, dependiendo de en qué

sistema operativo se esté ejecutando. Una vez en la versión final, esto no será necesario, ya que la dirección de acceso será la misma para todas las versiones de la aplicación.

A partir de aquí, se encargaría el servicio.

6.3.3.3 Seleccionar una página

Al seleccionar una página, como hemos visto en el apartado de diseño, hay diferentes resultados en función de la página que se seleccione. Primero teníamos el caso en el que la pagina seleccionada esté formada por una sola entidad, o solo una entidad que no contenga dependencias. Y por otro lado el caso en el que la página tuviese más de una entidad sin dependencias. En ambos casos el proceso es casi idéntico, primero solicitamos la página con su código desde `HomePage.cs`:

```
boton.Clicked += async (sender, e) =>
{
      string codigo = ((TagButton)sender).Tag;

      await Navigation.PushAsync(new SmartPage(codigo));
};
```

Esto crea una nueva `SmartPage`, que a su vez es una página de tipo `ContentPage`, en la cual tenemos una función inicializar, que pone en marcha el mecanismo de obtener la página:

```
private void Inicializar(string codigoPagina)
{
      Pagina = Negocio.GetPaginaSmart(codigoPagina).Result;
```

Desde la capa de negocio `bcFunc.cs` llamamos a la capa de datos para que nos proporcione la página:

```
public async Task<PaginaSmart> GetPaginaSmart(string codigo)
{
      return await Datos.GetPaginaSmart(codigo);
}
```

Desde la capa de datos llamamos a la función `CallGetService`(), que se encuentra en la clase `dcConn.cs` con el tipo de página y su código:

```
public async Task<PaginaSmart> GetPaginaSmart(string codigo)
        {

            PaginaSmart pagina = new PaginaSmart();

            string l = await Conexion.CallGetService("paginasmart/" +
codigo);
            //Correccion dobles barras \\ en json, doble codificación
            l = l.Replace(@"\", string.Empty).Trim(new char[] { '\"' });

            pagina = JsonConvert.DeserializeObject<PaginaSmart>(l);

            return pagina;
        }
```

Desde la función `CallGetService`() obtenemos el string JSON con la página:

```
public async Task<string> CallGetService(string urlApi)
        {
            using (var client = new HttpClient())
            {
                string url = "";
                  TRATAMIENTO DE LA URL
                var result = Task.Run(() => client.GetAsync(url)).Result;
                return await result.Content.ReadAsStringAsync();
            }
        }
```

Como podemos ver en la función `GetPaginaSmart`() de la capa de datos, en el momento que el cliente recibe el JSON y lo queremos de-serializar a objetos, el sistema no es capaz de hacerlo automáticamente puesto que la lista de entidades está definida para contener el objeto padre de ellas. Para ello nos vimos forzados a redefinir algunas de las funciones del `JsonConverter`. Creamos nuestra clase llamada `PaginaSmartConverter`, que extendía `JsonConverter` y modificamos la función `ReadJson`() para que satisficiese nuestras necesidades. De esta forma podemos recorrer la lista de entidades y dependiendo del tipo de-serializarlo a su objeto correcto. Todo esto está definido en la clase `PaginaSmart.cs` de la capa Sistema, que es común a ambos proyectos, tanto cliente como servicio:

```
public override object ReadJson(JsonReader reader, Type objectType, object
existingValue, JsonSerializer serializer)
{
      JObject jo = JObject.Load(reader);
       PaginaSmart pagina = new PaginaSmart();
       pagina.CodigoPagina = (string)jo["CodigoPagina"];
       pagina.Titulo = (string)jo["Titulo"];
       pagina.Ratio = (string)jo["Ratio"];
       pagina.Columnas = jo["Columnas"].ToObject<List<DefinicionColumna>>
      (serializer);
       pagina.Filas = jo["Filas"].ToObject<List<DefinicionFila>>
      (serializer);

       foreach (JObject obj in jo["Entidades"])
       {
            if ((string)obj["Clase"] == "WPartMultiselector")
                        pagina.Entidades.Add(
                        obj.ToObject<WPartMultiselector>(serializer));
            else if ((string)obj["Clase"] == "WPartConsulta")
                  pagina.Entidades.Add(obj.ToObject<WPartConsulta>(serializ
                  er));
            else if ((string)obj["Clase"] == "WPartConsultaFicha")
                  pagina.Entidades.Add(obj.ToObject<WPartConsultaFicha>
                  (serializer));
            else if ((string)obj["Clase"] == "WPartConsultaINI")
                  pagina.Entidades.Add(obj.ToObject<WPartConsultaINI>(seria
                  lizer));
            else if ((string)obj["Clase"] == "WPartGantt")
                  pagina.Entidades.Add(obj.ToObject<WPartGantt>(serializer)
                  );
            else if ((string)obj["Clase"] == "WPartGrafico")
                  pagina.Entidades.Add(obj.ToObject<WPartGrafico>(serialize
                  r));
            else if ((string)obj["Clase"] == "WPartIndicador")
                  pagina.Entidades.Add(obj.ToObject<WPartIndicador>(seriali
                  zer));
            else if ((string)obj["Clase"] == "WPartScheduler")
                  pagina.Entidades.Add(obj.ToObject<WPartScheduler>(seriali
                  zer));
      }
      return pagina;
}
```

Y a partir de aquí el servicio se encargaría de proporcionarnos la página.

Una vez hecho esto, que es común a todos los casos que se pueden dar al seleccionar una página, tenemos una pequeña bifurcación.

En el caso de que la página que hemos recuperado tenga más de una entidad no dependiente, procedemos a mostrar una lista con las diferentes entidades de las que consta:

```
if (listaEntidades.Count > 1)
{
      this.Title = Pagina.Titulo;

      ScrollView sv = new ScrollView();
      StackLayout sl = new StackLayout();
      sv.Content = sl;

      for (int i = 0; i < listaEntidades.Count; i++)
      {
            WPart wPart = listaEntidades[i];

            TagButton boton = new TagButton();
            boton.Text = wPart.Titulo;
            boton.Tag = wPart.Nombre;
            boton.Clicked += async (sender, e) =>
            {
                  string nombreWPart = ((TagButton)sender).Tag;
                  WPart wPart2 = listaEntidades.Where(item => item.Nombre
                  == nombreWPart).First();

                  await pcFunc.NavegarAWPart(wPart2, Pagina, this);
            };

            sl.Children.Add(boton);
      }

      Content = sv;
}
```

En el caso de que la página que hemos recuperado solo tenga una entidad o solo una no dependiente, procedemos a navegar directamente a la entidad en sí:

```
else
{
      WPart wPart = listaEntidades[0];
      this.Title = wPart.Titulo;
      if (wPart.Clase == "WPartGrafico")
            Content = new WPartVistaGrafico((WPartGrafico)wPart, "");
      else if (wPart.Clase == "WPartConsulta")
            Content = new WPartVistaConsulta((WPartConsulta)wPart, Pagina,
      "");
      else if (wPart.Clase == "WPartConsultaFicha")
            Content = new
      WPartVistaConsultaFicha((WPartConsultaFicha)wPart, "");
}
```

6.3.3.4 Seleccionar un Grafico

El funcionamiento de este proceso, como ya hemos visto en el diagrama de secuencia que hemos enseñado en el apartado anterior de mismo nombre, consiste en que, la página, nos proporciona la entidad, pero esto solo es su descriptor, el `WPart`. Aun no tenemos sus datos, por lo que es necesario recurrir al servicio REST de nuevo.

En este caso no utilizamos la capa de datos en el servicio REST, debido a que las funciones de obtención de datos de los `WPart` están proporcionadas por la empresa en unas DLL que ya hemos explicado en puntos anteriores.

Al igual que todos los datos, estos se serializan en formato JSON para ser enviados al cliente, este lo de-serializa y en la clase `VistaGrafico.cs`, transforma el diccionario de datos que teníamos en el gráfico.

La estructura de la clase es la siguiente:

```
public class WPartVistaGrafico : ContentView
    {
        private SfChart _Chart;
        private WPartGrafico _WPart;
        private Dictionary<string, ChartSeries> _ListaSeries;
        private bcFunc _Negocio;
        private string _Filtro;
```

Y el gráfico se genera de la siguiente manera:

```
public WPartVistaGrafico(WPartGrafico wPart, string filtro)
        {
            _Negocio = new bcFunc();

            _WPart = wPart;
            _Filtro = filtro;

            CrearGrafico();
            ObtenerDatos();

            Content = _Chart;
        }
```

Aquí podemos ver los dos puntos más importantes. Primero, dónde crea el gráfico, que es una función muy compleja y larga, y que se podrá ver en el código que se adjunta del proyecto. Básicamente se encarga de crear el objeto de tipo `SfChart` que nos proporciona SyncFusion, y asignar los parámetros que tenemos en nuestro `wPartGrafico`. Para la asignación de los datos de la serie con el gráfico, se usa un sistema de "binding", que consiste en enlazar automáticamente una serie de datos que tenemos con el componente `SfChart`, en este caso los datos del eje X y los datos del eje Y. Para ello es necesario crear un objeto que tenga los mismos nombres (`Categoria` y `Valor`) en el cual se encuentren los datos para poder hacer el "binding" correctamente.

```
if (serie.TipoGrafico == "Areas")
{
      AreaSeries areaSeries = new AreaSeries()
      {
            ...
            Color = pcFunc.ObtenerColorGrafico(serie.ColorGrafico),
            XBindingPath = "Categoria",
            YBindingPath = "Valor",
            EnableTooltip = true,
            ...
      };

      ...
}

else if (serie.TipoGrafico == "Areas Apiladas")

{

...

}
```

El segundo, es donde se obtienen los datos, que es lo que estamos representando en el diagrama de arriba. Cabe destacar que el `ItemSource` es el contexto de datos para el "binding". La llamada se realiza de la siguiente manera:

```
private void ObtenerDatos()
        {
          Dictionary<string, List<WPartGraficoDato>> datos =
          _Negocio.GetDatosWPartGrafico(_WPart).Result;

            foreach (var item in _ListaSeries)
            {
                item.Value.ItemsSource = datos[item.Key];
            }

        }
```

Llama a la capa de negocio para que le entregue los datos. Y a su vez, la capa de negocio llama a la capa de datos para que le entregue los datos:

```
public async Task<Dictionary<string, List<WPartGraficoDato>>>
GetDatosWPartGrafico(
                                                    WPartGrafico wPart)
{
      return await Datos.GetDatosWPartGrafico(wPart);
}
```

Y en la capa de datos, la clase `dcFunc.cs` llama a la clase `dcConn.cs` para que haga la conexión con el servicio REST:

```
public async Task<Dictionary<string, List<WPartGraficoDato>>>
GetDatosWPartGrafico(
                                                    WPartGrafico wPart)
{
      string l = await Conexion.CallPostService("WPartGrafico", wPart);
      //Correccion dobles barras \\ en json, doble codificación
      l = l.Replace(@"\", string.Empty).Trim(new char[] { '\"' });

      return JsonConvert.DeserializeObject<Dictionary<string,
      List<WPartGraficoDato>>>(l);
}
```

`Conexion` es una instancia de la clase `dcConn`, en la que llamamos al método `CallPostService`(), con la primera parte de la URL que usaremos, y el dato, que es el `WPart`:

```
public async Task<string> CallPostService(string urlApi, object
wPart_filtro)
{
      using (var client = new HttpClient())
      {
            HttpContent contentPost = new
            StringContent(JsonConvert.SerializeObject(wPart_filtro)
                  , Encoding.UTF8, application/json");

      return await client.PostAsync(new Uri(url), contentPost)
            .ContinueWith((postTask) => postTask.Result
            .EnsureSuccessStatusCode())
            .Result.Content.ReadAsStringAsync();
      }
}
```

Una vez llegados aquí, del resto del trabajo se encarga la parte del servicio REST, como ya se ha visto anteriormente.

6.3.3.5 Seleccionar una Consulta

Al igual que en el gráfico, solo tenemos la descripción del `WPartConsulta`, no tenemos sus datos aun, por lo que es necesario recurrir al servicio REST de nuevo.

En este caso tampoco utilizaremos la capa de datos en el servicio REST, debido a que las funciones de obtención de datos de los `WPart` están proporcionadas por la empresa en unas DLL que ya hemos explicado en puntos anteriores.

Al igual que todos los datos, estos se serializan en formato JSON para ser enviados al cliente, este lo de-serializa y en la clase `WPartVistaConsulta`, transforma la lista de datos que teníamos en la consulta.

La estructura de la clase es la siguiente:

```
class WPartVistaConsulta : ContentView
    {
        private ListView _ListView;
        private WPartConsulta _WPart;
        private PaginaSmart _Pagina;
        private bcFunc _Negocio;
        private string _Filtro;
```

Y la consulta se genera de la siguiente manera:

```
public WPartVistaConsulta(WPartConsulta wPart, PaginaSmart pagina, string
filtro)
{
      _Negocio = new bcFunc();

      _WPart = wPart;
      _Pagina = pagina;
      _Filtro = filtro;

      CrearConsulta();
      ObtenerDatos();

      Content = _ListView;
}
```

Aquí podemos ver los dos puntos más importantes, primero, dónde crea la consulta, que es una función muy larga, y que se podrá ver en el código que se adjunta en el proyecto. Básicamente, se encarga de crear el objeto de tipo `ListView` que nos proporciona Xamarin y asignar los datos que tenemos en nuestra lista, haciendo uso también del método de “binding”.

El segundo, es donde se obtienen los datos, que es lo que estamos representando en el diagrama de arriba. La llamada se realiza de la siguiente manera:

```
private void ObtenerDatos()
{
	_ListView.ItemsSource = _Negocio.GetDatosWPartConsulta(_WPart,
_Filtro).Result;
}
```

Llama a la capa de negocio para que le entregue los datos. Y a su vez, la capa de negocio llama a la capa de datos para que le entregue los datos:

```
public async Task<List<Dictionary<string, string>>> GetDatosWPartConsulta(
                                        WPartConsulta wPart, string
                                        filtro)
{
	return await Datos.GetDatosWPartConsulta(wPart, filtro);
}
```

Y en la capa de datos, la clase `dcFunc.cs` llama a la clase `dcConn.cs` para que haga la `conexión` con el servicio REST:

```
public async Task<List<Dictionary<string, string>>>
GetDatosWPartConsulta(WPartConsulta _wPart, string _filtro)
{
	WPartConsultaConFiltro wPart_filtro = new WPartConsultaConFiltro();
	wPart_filtro.WPart = _wPart;
	wPart_filtro.Filtro = _filtro;

	string l = await Conexion.CallPostService("WPartConsulta",
wPart_filtro);
	//Correccion dobles barras \\ en json, doble codificación
	l = l.Replace(@"\", string.Empty).Trim(new char[] { '\"' });

	return JsonConvert.DeserializeObject<List<Dictionary<string,
string>>>(l);
}
```

`Conexion` es una instancia de la clase `dcConn.cs`, en la que llamamos al método `CallPostService`(), con la primera parte de la URL que usaremos, y el dato, que es el `WPart`:

```
public async Task<string> CallPostService(string urlApi, object
wPart_filtro)
{
      using (var client = new HttpClient())
      {
            HttpContent contentPost = new
            StringContent(JsonConvert.SerializeObject(wPart_filtro),
      Encoding.UTF8,
                  "application/json");

            return await client.PostAsync(new Uri(url),
      contentPost).ContinueWith(
            (postTask) => postTask.Result.EnsureSuccessStatusCode())
            .Result.Content.ReadAsStringAsync();
            }
        }
```

Una vez llegados aquí, del resto del trabajo se encarga la parte del servicio REST, como ya se ha visto anteriormente.

6.3.3.6 Seleccionar una Consulta Ficha

AL igual que en el gráfico y la consulta, solo tenemos la descripción del `WPartConsultaFicha`, no tenemos sus datos aun, por lo que es necesario recurrir al servicio REST de nuevo.

En este caso tampoco utilizamos la capa de datos en el servicio REST, debido a que las funciones de obtención de datos de los `WPart` están proporcionadas por la empresa en unas DLL que ya hemos explicado en puntos anteriores.

Al igual que todos los datos, estos se serializan en formato JSON para ser enviados al cliente, este lo de-serializa y en la clase `WPartVistaConsultaFicha`.cs, transforma el diccionario de datos que teníamos en la ficha.

La estructura de la clase es la siguiente:

```
public class WPartVistaConsultaFicha : ContentView
    {
        private ScrollView _SV;
        private StackLayout _SL;
        private WPartConsultaFicha _WPart;
        private bcFunc _Negocio;
        private string _Filtro;
```

Y la ficha se genera de la siguiente manera:

```
public WPartVistaConsultaFicha(WPartConsultaFicha wPart, string filtro)
{
	_Negocio = new bcFunc();

	_WPart = wPart;
	_Filtro = filtro;

	CrearConsulta();
	ObtenerDatos();

	Content = _SV;
}
```

Aquí podemos ver los dos puntos más importantes, primero, dónde crea la ficha:

```
private void CrearConsulta()
{
	_SV = new ScrollView();
	_SL = new StackLayout();

	Dictionary<string, ConfiguracionCampoConsultaFicha>
	listaOrdenadaCampos = new Dictionary<string,
	ConfiguracionCampoConsultaFicha>();

	for (int i = 0; i < _WPart.TitulosCampos.Count; i++)
	{
		ConfiguracionCampoConsultaFicha cfg = _WPart.TitulosCampos[i];
		listaOrdenadaCampos.Add(cfg.NombreCampo, cfg);
	}

	for (int i = 0; i < _WPart.Campos.Count; i++)
	{
		if (!listaOrdenadaCampos.ContainsKey(_WPart.Campos[i]))
			listaOrdenadaCampos.Add(_WPart.Campos[i], null);
	}
```

Creamos un `ScrollView` donde meter la lista, que es un `StackLayout`. Creamos un diccionario con el tipo de datos `ConfiguracionCampoConsultaFicha` que es la siguiente clase:

```
public class ConfiguracionCampoConsultaFicha
{
	protected string _NombreCampo;
	protected string _Titulo; // posición 0
```

Y procedemos a añadir los títulos.

```
foreach (var item in listaOrdenadaCampos)
{
      ...

      string titulo = "";
      if (nombreCampo.Contains("."))
            titulo = nombreCampo.GetToken(1, ".");
```

Luego utilizamos un bucle para dar formato a la lista, ya que no viene como nosotros deseamos. Lo que hacemos es reemplazar varios símbolos como "_" por "."

```
if (nombreCampo.Contains("."))
      nombreCampo = nombreCampo.Replace(".", "_");

...

_SL.Children.Add(valorCampo);
```

El segundo, es donde se obtienen los datos, que es lo que estamos representando en el diagrama de arriba. La llamada se realiza de la siguiente manera:

```
private void ObtenerDatos()
{
      _SL.BindingContext = _Negocio.GetDatosWPartConsultaFicha(_WPart,
      _Filtro).Result;
}
```

Llama a la capa de negocio para que le entregue los datos. Y a su vez, la capa de negocio llama a la capa de datos para que le entregue los datos:

```
public async Task<Dictionary<string, string>> GetDatosWPartConsultaFicha(
                                    WPartConsultaFicha wPart, string
                                    filtro)
{
      return await Datos.GetDatosWPartConsultaFicha(wPart, filtro);
}
```

Y en la capa de datos, la clase `dcFunc.cs` llama a la clase `dcConn.cs` para que haga la conexión con el servicio REST:

```
public async Task<Dictionary<string, string>> GetDatosWPartConsultaFicha
(WPartConsultaFicha _wPart, string _filtro)
{
      WPartConsultaFichaConFiltro wPart_filtro = new
      WPartConsultaFichaConFiltro();
      wPart_filtro.WPart = _wPart;
      wPart_filtro.Filtro = _filtro;

      string l = await Conexion.CallPostService("WPartConsultaFicha",
      wPart_filtro);
      //Correccion dobles barras \\ en json, doble codificación
      l = l.Replace(@"\", string.Empty).Trim(new char[] { '\"' });

      return JsonConvert.DeserializeObject<Dictionary<string, string>>(l);
}
```

Conexion es una instancia de la clase dcConn.cs, en la que llamamos al método CallPostService(), con la primera parte de la URL que usaremos y el dato, que es el WPart:

```
public async Task<string> CallPostService(string urlApi, object
wPart_filtro)
{
      using (var client = new HttpClient())
      {
            HttpContent contentPost = new
            StringContent(JsonConvert.SerializeObject(wPart_filtro),
      Encoding.UTF8,
                  "application/json");

            return await client.PostAsync(new Uri(url),
      contentPost).ContinueWith(
            (postTask) => postTask.Result.EnsureSuccessStatusCode())
            .Result.Content.ReadAsStringAsync();
      }
}
```

Una vez llegados aquí, el resto del trabajo se encarga la parte del servicio REST, como ya se ha visto anteriormente.

6.3.3.7 El caso de que una consulta tenga hijos

Como hemos visto en el diagrama, puede darse el caso de que una entidad dependa de otra, en ese caso, esa entidad recibe un filtro, que está en la definición de la entidad padre.

El filtro permite delimitar los datos que mostrará la entidad dependiente.

Y a la hora de seleccionar una de las entidades que son dependientes, a la hora de recuperar los datos es necesario que enviemos la entidad, al igual que en el caso de las que no son dependientes, pero, además, tenemos que enviar el filtro. Para ello, enviamos un objeto en el cual tenemos ambos elementos, la entidad y el filtro. Su estructura es la siguiente:

```
public class WPartConsultaFichaConFiltro
{
	protected string _Filtro = "";
	protected WPartConsultaFicha _WPart;

	public string Filtro
	{
		get { return _Filtro; }
		set { _Filtro = value; }
	}
	public WPartConsultaFicha WPart
	{
		get { return _WPart; }
		set { _WPart = value; }
	}
}
```

En nuestra aplicación solo sucede con las entidades de tipo `WPartConsultaFicha`, pero en un futuro cuando se implemente por completo, será necesario el uso de este tipo de estructuras.

6.3.4 Pruebas

En este capítulo se recogen las pruebas realizadas sobre los módulos, las pruebas de integración y las pruebas de validación.

6.3.4.1 Pruebas unitarias

El objetivo principal de estas pruebas es validar el comportamiento de las funciones que componen cada módulo.

Para realizar estas pruebas hemos diseñado unos escenarios en los que se pueda verificar que la aplicación tiene una calidad aceptable.

6.3.4.2 Diseño de los casos de prueba

Los casos de prueba aquí mencionados son los relacionados con las pruebas de integración del sistema. Los casos de prueba en que se han dividido el plan de pruebas pueden agruparse en dos tipos:

- Pruebas de ejecución correctas
- Pruebas con error

Las pruebas de ejecuciones correctas tienen como objetivo verificar las funcionalidades del sistema, es decir que, dado un caso, cumpla el comportamiento esperado.

Las pruebas de ejecuciones con error tienen como objetivo verificar el correcto tratamiento de los errores por parte de la aplicación, es decir, forzando la aplicación a que cometa algún error.

Pruebas de ejecución correctas

La aplicación tiene unos casos de uso muy concretos, por lo que nos hemos asegurado de que cada uno funcione correctamente.

- Solicitar lista de páginas: Funciona correctamente.
- Seleccionar una página: Funciona correctamente.
- Seleccionar una de las entidades de una página: Funciona correctamente a excepción de una entidad.
- Ver la línea completa: Funciona correctamente.

Pruebas de ejecución con error

Dado que en esta aplicación no se espera ningún dato de entrada por parte del usuario, todo queda muy predefinido, lo que no da lugar a errores por parte de los usuarios que nosotros debamos de tratar.

El único error que debemos controlar es si la hay conexión a internet o no.

6.3.4.3 Pruebas de validación

Una vez realizadas las pruebas de integración, se han realizado las pruebas de validación. El objetivo principal de estas pruebas, es comprobar que se cumplen los requisitos funcionales y el rendimiento que se ha definido en un principio con la empresa. Para validar el sistema se ha utilizado la técnica de la caja negra, empleando como criterio de validación la aceptación del sistema por parte de la empresa

7 CONFIGURACIÓN DE LA BASE DE DATOS

Aquí es donde se realizan todas las conexiones con la base de datos que nos han proporcionado, en este caso hemos utilizado Entity Framewok:

''Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos. Los arquitectos y programadores de aplicaciones orientadas a datos se han enfrentado a la necesidad de lograr dos objetivos muy diferentes. Deben modelar las entidades, las relaciones y la lógica de los problemas empresariales que resuelven, y también deben trabajar con los motores de datos que se usan para almacenar y recuperar los datos. Los datos pueden abarcar varios sistemas de almacenamiento, cada uno con sus propios protocolos; incluso las aplicaciones que funcionan con un único sistema de almacenamiento deben equilibrar los requisitos del sistema de almacenamiento con respecto a los requisitos de escribir un código de aplicación eficaz y fácil de mantener.

Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos del dominio, como clientes y direcciones de cliente, sin tener que preocuparse por las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con Entity Framework, los desarrolladores pueden trabajar en un nivel mayor de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código que en las aplicaciones tradicionales. Dado que Entity Framework es un componente de .NET Framework, las aplicaciones de Entity Framework se pueden ejecutar en cualquier equipo en el que esté instalado .NET Framework a partir de la versión 3.5 SP1.''

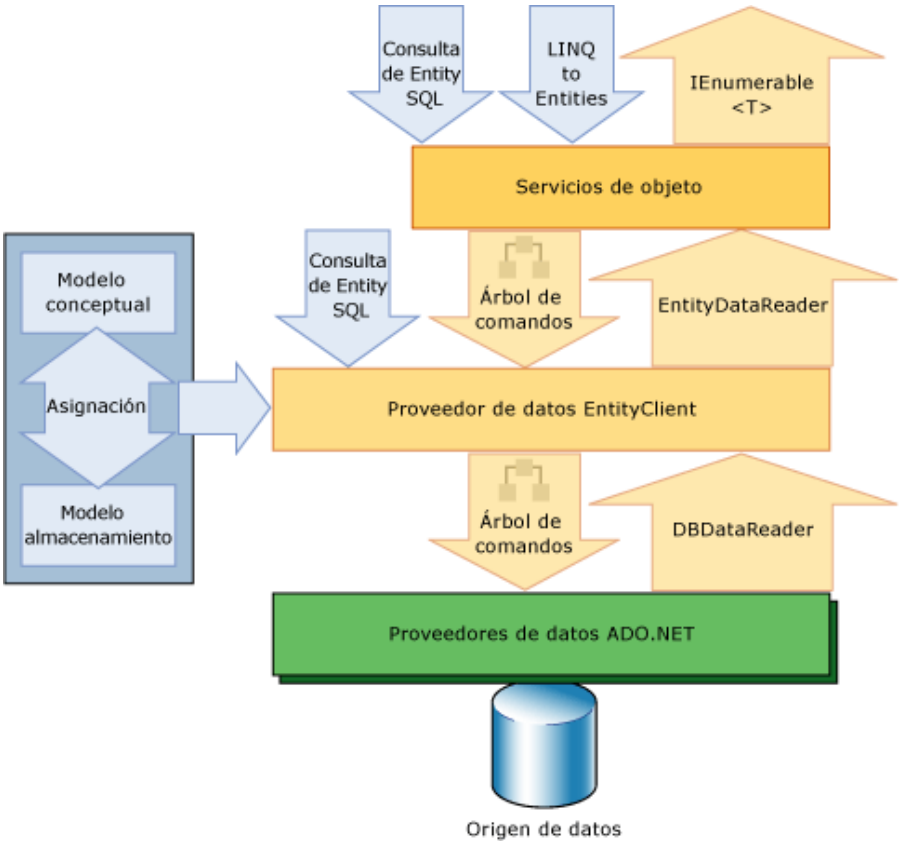


Imagen 7.1. Estructura Entity Framework

Con esto, lo que hemos hecho ha sido crear un modelo conceptual de los datos, asignar los objetos a los datos y luego obtener el acceso a los datos de la entidad. La manera que tenemos de manejar los datos es, en este caso, con LinQ, un lenguaje similar a SQL.

7.1 PASOS PARA LA CREACIÓN DE LA BASE DE DATOS

Aquí explicaremos paso a paso como enlazar la base de datos con nuestro proyecto usando Entity Framework.

7.1.1 Crear la aplicación

Los pasos que deberemos seguir son los siguientes:

1. Abrimos **Visual Studio**
2. **Archivo** -> **Nuevo** -> **Proyecto**
3. Seleccionamos **Windows** en el menú de la izquierda
4. Nombramos el proyecto como **CDatos**, de capa de datos.
5. Seleccionamos **Aceptar**

7.1.2 Crear el modelo

Las clases están definidas en un proyecto a parte, que es el de CSistema, donde tenemos todos los objetos ya definidos.

7.1.3 Crear el contexto

Utilizaremos el Asistente de **Entity Data Model** para generar un archivo .edmx que contiene un modelo conceptual, un modelo de almacenamiento e información de asignación. Este archivo define un conjunto de asignaciones 1:1 entre entidades y tablas para el modelo conceptual de eSmartCoreClient y su base de datos.

Para agregar la plantilla de elementos de Entity Data Model de ADO.NET

1. Seleccionamos el proyecto **CDatos** en el **Explorador de soluciones**, hacemos clic con el botón secundario, elegimos **Agregar** y hacemos clic en **Nuevo elemento**.
2. Seleccionamos **Entity Data Model de ADO.NET** en el recuadro **Plantillas**.
3. Escribimos **WebCRMG.edmx** como nombre del modelo y hacemos clic en **Agregar**.
4. Se muestra la página inicial del asistente de **Entity Data Model**.

Para generar el archivo .edmx

1. Seleccionamos Generar desde base de datos en el cuadro de diálogo **Elegir contenido del modelo**. Después, hacemos clic en **Siguiente**.
2. Hacemos clic en el botón **Nueva conexión**.
3. En el cuadro de diálogo **Elegir origen de datos**, seleccionamos su origen de datos y, a continuación, hacemos clic en **Continuar**.
4. En el cuadro de diálogo **Propiedades de conexión**, escribimos nuestro nombre de servidor, seleccionamos el método de autenticación, escribimos **WebCRMG** como nombre de la base de datos y, a continuación, hacemos clic en **Aceptar**.

   El cuadro de diálogo **Elegir la conexión de datos** se actualiza con la configuración de la conexión de base de datos.

5. Nos aseguramos de que la opción **Guardar configuración de conexión de entidad en App.Config como:** está seleccionada y el valor está establecido en **WebCRMGEntities**. Después, hacemos clic en **Siguiente**.

   Aparece el cuadro de diálogo **Elija los objetos de base de datos**.

6. Seleccionamos todas las tablas y procedimientos almacenados y nos aseguramos de que el valor de **Espacio de nombres del modelo** es **WebCRMG**.
7. Nos aseguramos de que las opciones **Poner en plural o en singular los nombres de objeto generados** e **Incluir columnas de clave externa en el modelo** están seleccionadas.

El asistente realiza las siguientes acciones:

- Agrega referencias a los ensamblados System.Data.Entity, System.Security y System.Runtime.Serialization.
- Genera el archivo WebCRMG.edm que define el modelo conceptual, el modelo de almacenamiento y la asignación entre los dos modelos.
- Crea un archivo de código de nivel de objetos que contiene las clases que se generaron según el modelo conceptual. Para ver el archivo de código de nivel de objetos, expandimos el nodo del archivo .edmx en el Explorador de soluciones.
- Crea un archivo App.config

Para ver el archivo .edmx en el ADO.NET Entity Data Model Designer

En el **Explorador de soluciones**, hacemos doble clic en el archivo WebCRMG.edmx De este modo se muestra el modelo WebCRMG en la ventana del diseñador de Entity Data Model de ADO.NET, según se muestra en el diagrama siguiente.

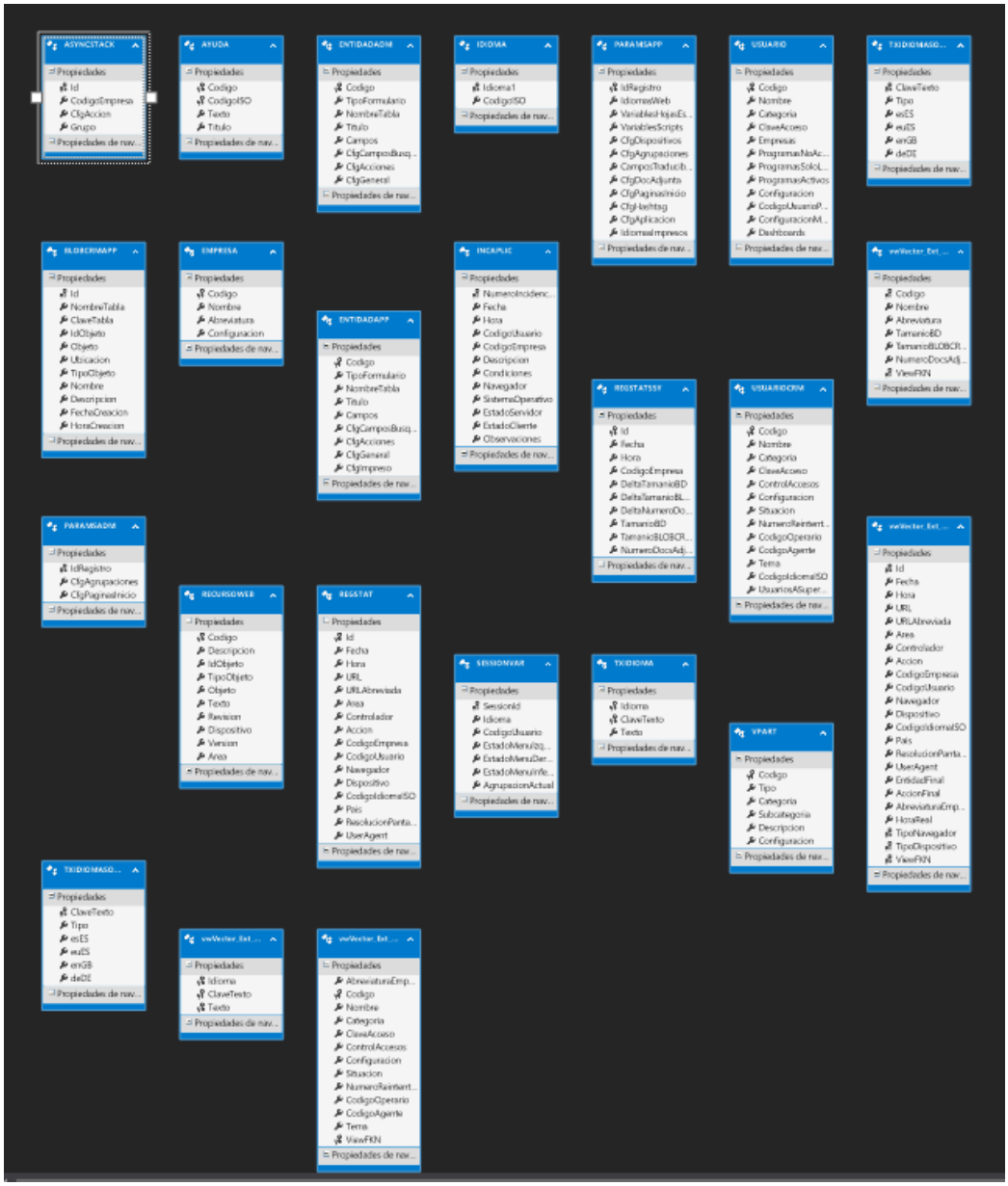


Imagen 7.1.3.1. Tablas base de datos

Aunque en nuestro proyecto, la única tabla de la que debemos preocuparnos es de la tabla VPART, que es la que usaremos en nuestra aplicación demo:

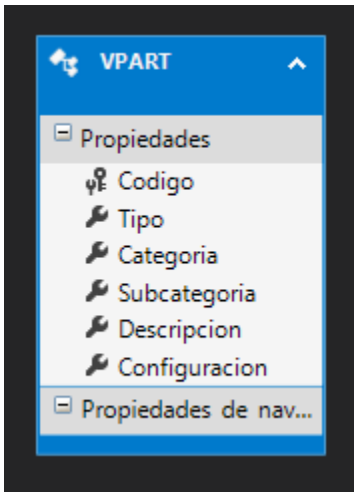


Imagen 7.1.3.2. Tabla VPART

Para consultar los datos, en las clases que se vayan a implementar las funciones, habrá que invocar al contexto para poder acceder a los datos modelados en forma de objetos.

8 GESTIÓN DEL PROYECTO

En este capítulo hablaremos sobre la gestión del proyecto y la ejecución del mismo. En los siguientes puntos se mostrarán la descomposición de tareas, fechas de realización, hitos definidos, el esfuerzo y el seguimiento del proyecto,

El periodo que se refleja en este documento comprende desde el 9 de septiembre de 2016 hasta el 18 de enero de 2017. Este es el tiempo real que se ha tardado en desarrollar el proyecto.

Los datos que se muestran son los recogidos en al inicio del proyecto y los tomados durante el desarrollo del proyecto, de manera que se puede realizar una comparativa entre ambos viendo la desviación sufrida respecto a la estimación inicial.

Las tareas de este proyecto han sido realizadas en el orden que se indica en la imagen que se muestra a continuación, teniendo en cuenta que para empezar una de ellas debía haber terminado la actividad previa. Por lo tanto, el modelo a seguir en este proyecto ha sido en “Cascada”. Al finalizar cada una de las etapas se han probado exhaustivamente y han sido supervisadas por el tutor de la empresa. Después de esta revisión, los puntos que han sido aprobados han pasado a formar parte de la aplicación final.

El siguiente esquema muestra las fases en las que se ha dividido el proyecto.

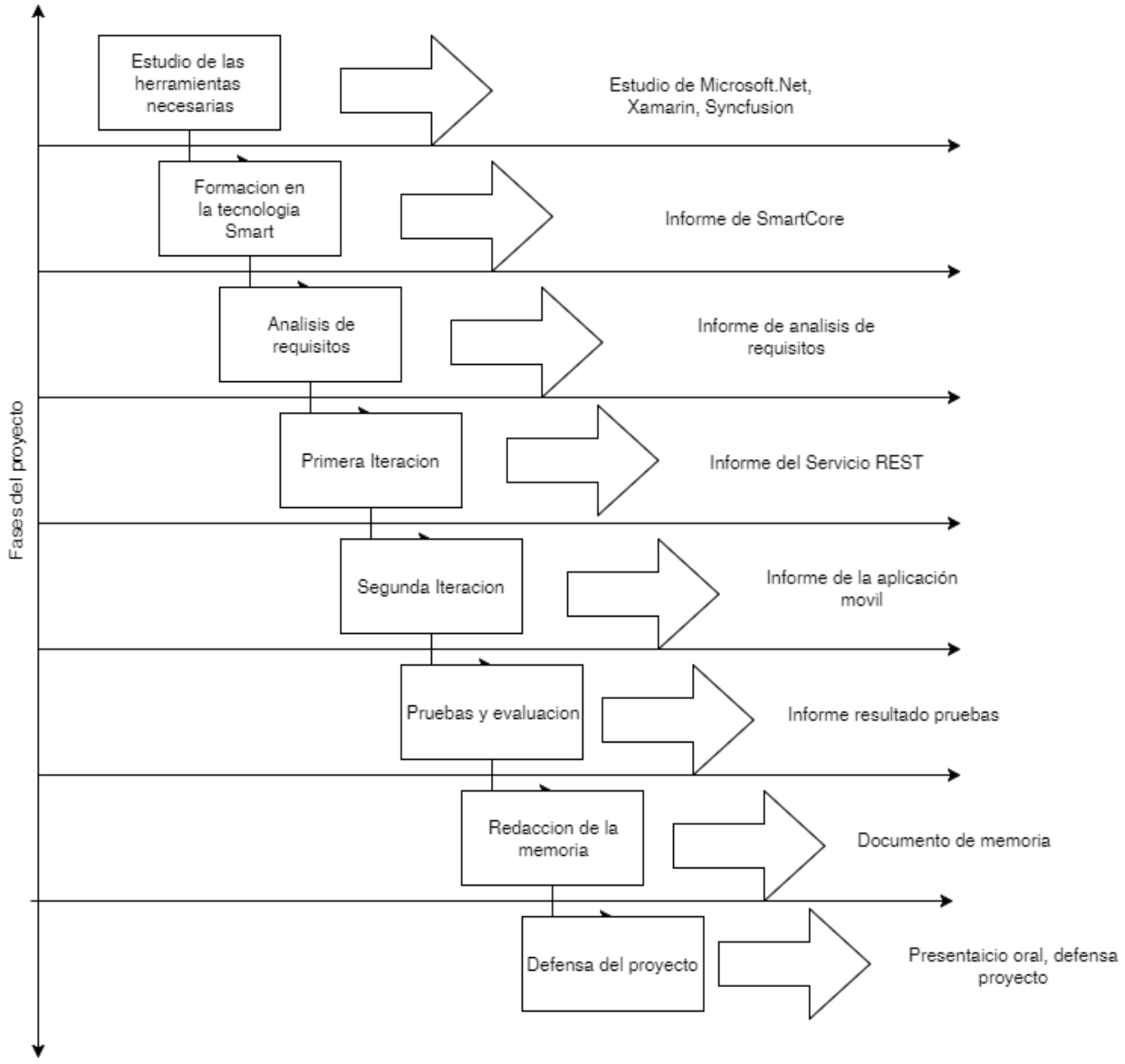


Imagen 8.1. Fases del proyecto.

8.1 ETD

En la siguiente tabla se puede ver para cada tarea la diferencia entre tiempo estimado y el tiempo real invertido en realizarlas.

Tarea	Estimado	Real
Estudio herramientas necesarias	12h	12h
Formación en tecnología Smart	25h	12h
Análisis de requisitos	15h	3h
Primera iteración	65h	40h
Segunda iteración	75h	120h
Pruebas y evaluación	15h	15h
Redacción de la memoria	80h	100h
Defensa del proyecto	15h	15h
Total	302h	317h

Tabla 8.1.1. Tabla de comparación de horas

8.2 DIAGRAMA DE GANTT

Con el fin de presentar las fases que se han seguido en el desarrollo del proyecto y el esfuerzo empleado en cada una de ellas, se muestran a continuación todos los diagramas de Gantt que se han ido haciendo a medida que re planificaba el proyecto por los distintos retrasos. El diagrama representa la planificación inicial del proyecto, mientras que los posteriores van mostrando las distintas re-planificaciones con sus supuestos retrasos en las fechas, y los motivos por los cuales se han producido.

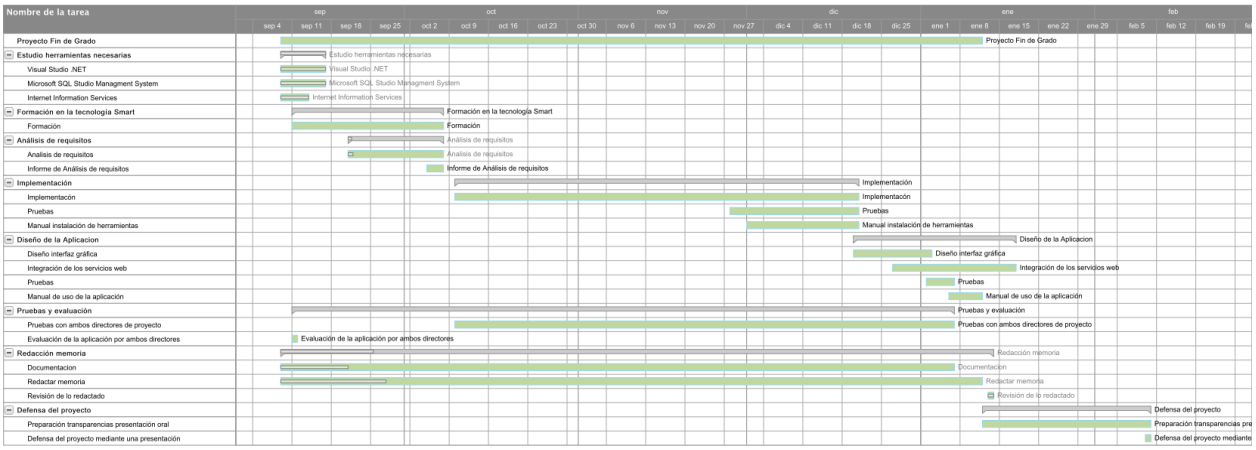


Diagrama Gantt 8.2.1. Debido a que la imagen no se ve muy bien, dejamos un link al pdf original.

Motivos para la segunda planificación: Debido a que el formato de desarrollo del proyecto se mejoró, se volvieron a definir los pasos a seguir, separando por un lado en particular, el desarrollo del servicio REST y por otro lado el desarrollo de la aplicación móvil, cada uno con sus respectivos análisis, diseños, implementaciones y pruebas. A continuación, se muestra la segunda planificación en la que se pueden ver los cambios citados:

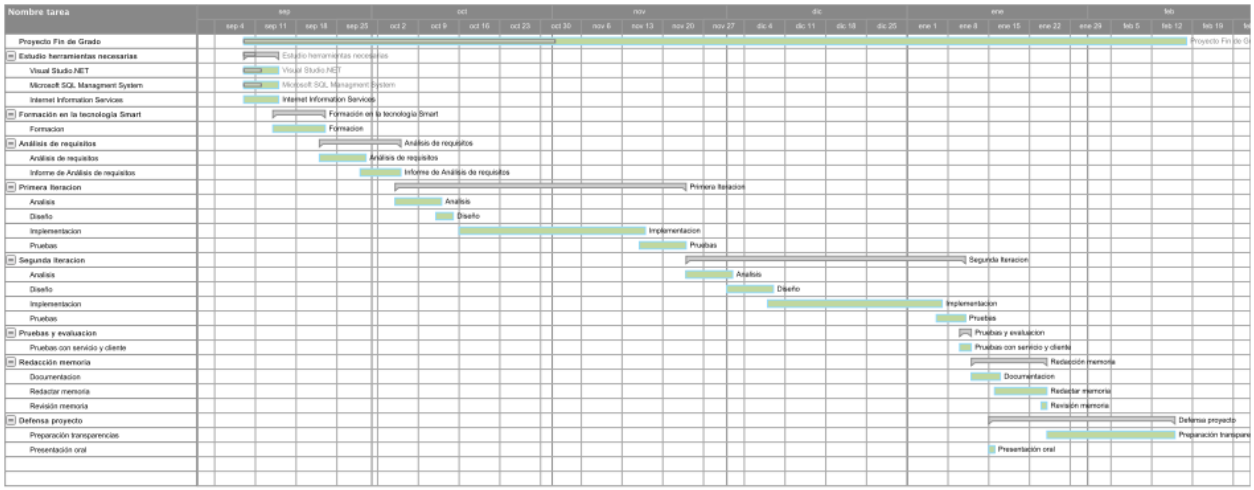


Diagrama Gantt 8.2.2. Debido a que la imagen no se ve muy bien, dejamos un link al pdf original

Motivos para la tercera planificación: debido a que el periodo que teníamos previsto para el desarrollo del servicio REST nos llevó menos de lo esperado, tuvimos que volver a definir la duración del proyecto y el tiempo que le dedicaríamos a el desarrollo de la aplicación móvil. Así, el periodo se acortó en una semana. La causa fue que se dedicaron más horas de las previstas esos días, y que contábamos con más código compatible del que nos esperábamos, por lo que hubo menos trabajo. A continuación, se muestra la tercera planificación en la que se pueden ver los cambios citados:

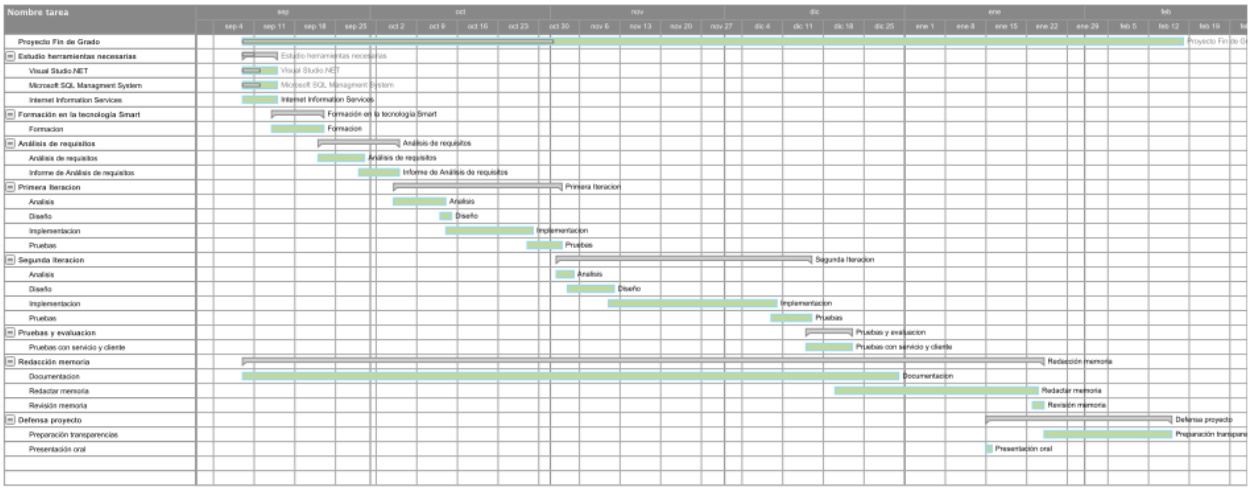


Diagrama Gantt 8.2.3. Debido a que la imagen no se ve muy bien, dejamos un link al pdf original

Motivos para la cuarta planificación: una vez desarrollada la aplicación móvil, el aprendizaje de Xamarin nos tomó más tiempo del que teníamos previsto. Hasta que tuvimos todo en marcha, nos retrasamos una semana, la misma que habíamos ganado anteriormente, por lo que aun seguíamos en plazo. El proyecto volvió a tener la misma duración que al principio. La causa fue que falté algunos días por enfermedad, y la falta de documentación en la red sobre Xamarin. A continuación, se muestra la cuarta planificación en la que se pueden ver los cambios citados:

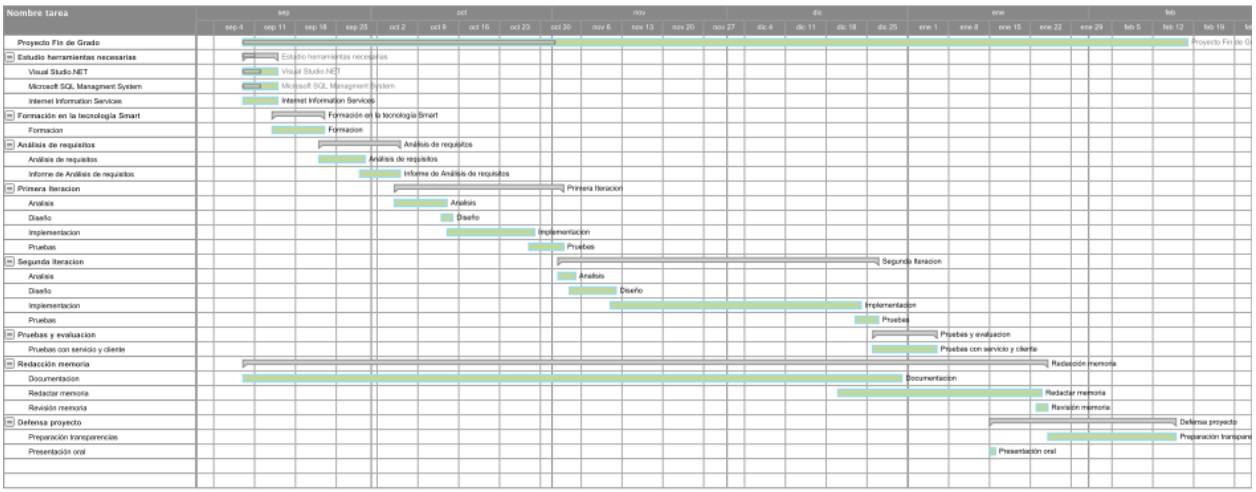


Diagrama Gantt 8.2.4. Debido a que la imagen no se ve muy bien, dejamos un link al pdf original

Motivos para la quinta planificación: un rediseño en la documentación que no esperábamos, nos retrasó la fecha del proyecto en dos semanas. El contenido en sí de la documentación era correcto, pero no estaba bien estructurado y daba lugar a confusiones, por lo que, aconsejados por el director del proyecto, se estructuro según sus indicaciones. Esto nos dejó la fecha de fin del proyecto en el 19-01-17. A continuación se muestra la quinta planificación en la que se pueden ver los cambios citados:

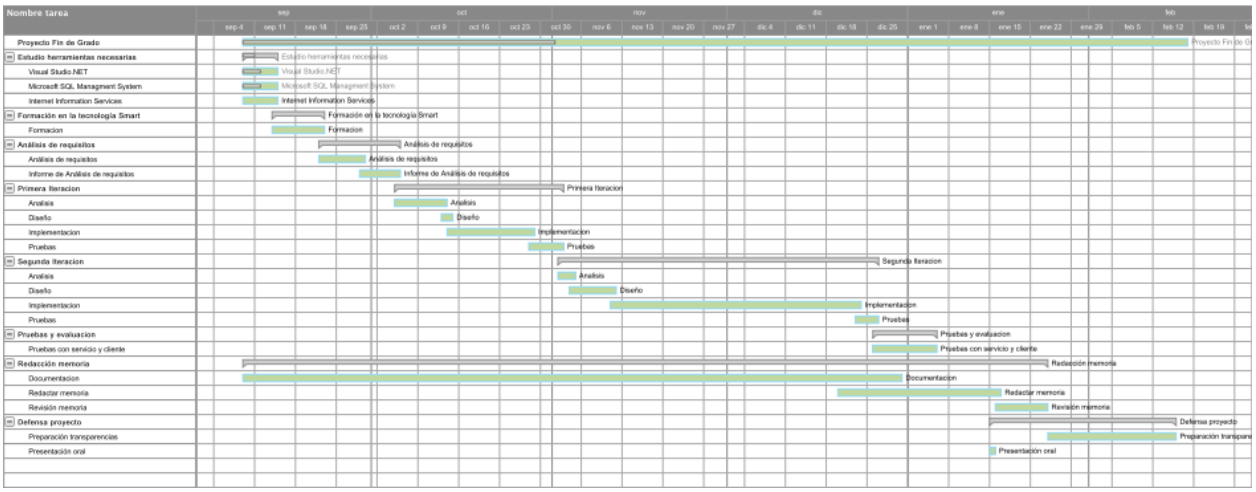


Diagrama Gantt 8.2.5. Debido a que la imagen no se ve muy bien, dejamos un link al pdf original

8.3 COMPARACIÓN, ESTIMACIÓN PLANIFICADA Y DEDICACIÓN REAL

Un proceso esencial a la hora de desarrollar un proyecto es llevar un continuo seguimiento del mismo, para que no nos coja por sorpresa la aparición de algún problema o retraso. A continuación, se muestra un gráfico que representa los datos comparativos entre los días estimados en la planificación y los días que realmente se han invertido en el desarrollo del proyecto.

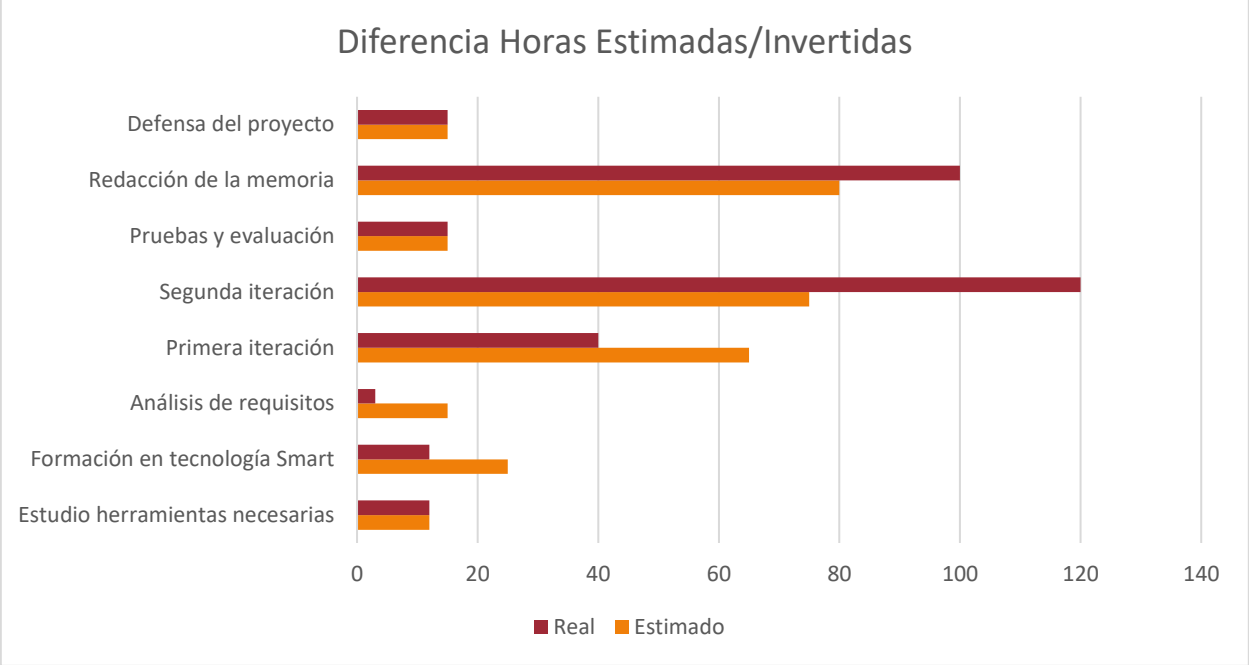
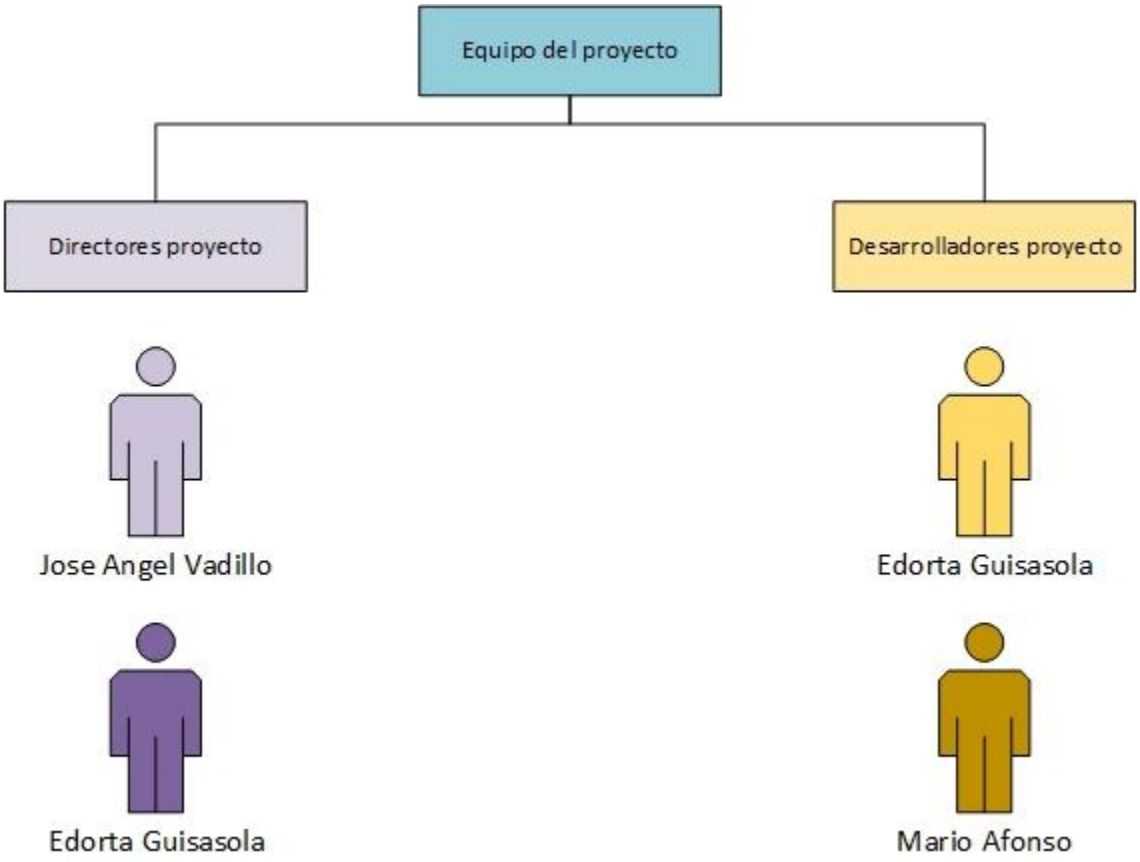


Grafico 8.3.1. Comparación horas estimadas con las invertidas

8.4 EQUIPO DEL PROYECTO

Estos son los miembros que han formado parte del equipo del proyecto:



8.4.1 Comunicación entre los miembros del equipo

La comunicación entre los miembros del equipo forma un papel muy importante a la hora de desarrollar un proyecto, y de ella depende en gran medida el éxito del mismo. Desde el inicio del proyecto se ha mantenido buena comunicación entre todos los miembros. El director del proyecto en la empresa, ha mantenido un riguroso seguimiento del desarrollo del proyecto, y una vez finalizado, con cada gran cambio en la documentación, o con dudas con respecto a esta se ha informado al director del proyecto en la universidad. En las reuniones diarias con el director del proyecto en la empresa se han presentado las tareas realizadas y se han discutido las posibles alternativas en la realización del proyecto hasta construir la aplicación final.

8.5 SEGUIMIENTO DEL PROYECTO

Un proceso esencial a la hora de desarrollar un proyecto es llevar un continuo seguimiento del mismo, para que no nos coja por sorpresa la aparición de algún problema o retraso. Durante el desarrollo de este proyecto se han realizado reuniones casi a diario para llevar a cabo este seguimiento, tratando en cada una de ellas los temas de interés y encontrando soluciones a los problemas que han surgido. Además de un control horario muy estricto.

9 CONCLUSIONES

Una vez finalizado el proyecto las conclusiones obtenidas son satisfactorias. Estas se enumeran a continuación.

En lo que se refiere a los objetivos del proyecto, no estoy plenamente satisfecho con el trabajo realizado, ya que los objetivos previstos en un principio no se han cumplido del todo. Sí se ha obtenido el producto deseado, pero hay aspectos en los cuales me habría gustado poder dar un repaso más, sobre todo en el tema visual de la aplicación móvil.

El producto final obtenido reúne todas las expectativas que tenía la empresa, al menos del lado funcional. Aunque es cierto que este proyecto se seguirá desarrollando en la empresa, la base que se ha hecho es sólida y con mucho potencial. Y la aplicación de demostración que hemos desarrollado prueba que el servicio desarrollado funciona como la empresa quería.

El objetivo inicial, era trasladar la tecnología que tenían a servicios web consumibles desde cualquier aplicación, y eso se ha conseguido.

Durante todo el proyecto los plazos previstos se han ido cumpliendo, habiendo fases terminadas según lo previsto, otras con antelación y otras con cierto retraso, obteniendo así una desviación media. Esto ha sido debido a mi poca experiencia a la hora de realizar proyectos y a una planificación inicial no muy buena. Las reuniones diarias han sido esenciales para seguir la senda marcada correctamente y de manera más eficaz.

Tras la finalización de este Proyecto Fin de Carrera he obtenido una visión mucho más completa del proceso de desarrollo y gestión de un proyecto de Ingeniería Informática y espero que los conocimientos y la experiencia adquiridos me sirvan como base para incorporarme al mundo laboral.

10 BIBLIOGRAFÍA Y ANEXOS

Para la realización de este proyecto hemos consultado gran cantidad de fuentes, entre las que se encuentran:

[1] StackOverflow. http://es.stackoverflow.com/

[2] Xamarin. https://developer.xamarin.com/

[3] Syncfusion. https://www.syncfusion.com/products/

[4] SmartSheet. https://app.smartsheet.com/

[5] Almacenamiento de datos del proyecto. https://onedrive.live.com, https://drive.google.com

[6] Msdn. https://msdn.microsoft.com/

[7] Wikipedia. https://es.wikipedia.org/

[8] Google. https://www.google.com/

[9] Azure. https://azure.microsoft.com/

10.1 ANEXO A: INFORMES MENSUALES

INFORME DEL MES DE SEPTIEMBRE

A día 9 de octubre del 2016 el estado del proyecto es el siguiente:

- Redacción de la memoria:
    - He realizado el DOP en el periodo del 13-09-16 al 3-10-16, invirtiendo para ello 15h
    - Formación herramientas necesarias:
        - He finalizado el estudio de Visual Studio, Microsoft SQL Management System y la herramienta Microsoft Internet Information Services. Todo esto me ha llevado un total de 12h.
    - Formación en la tecnología "Smart":
        - La empresa me ha instruido completamente en su tecnología, esto nos ha tomado 12h.
    - Análisis de requisitos:
        - He finalizado el análisis de requisitos para el proyecto. Con ayuda del director del proyecto en la empresa hemos concluido esta tarea en 3h.

- Primera iteración:
    - De la primera Iteración hemos concluido con el análisis de esta y con el diseño. Este último no nos ha llevado mucho tiempo, debido a que la mayoría ya estaba diseñado y establecido por la empresa. Un total de 5h.

Conclusiones: Este es el primer informe, que abarca el periodo del 09-09-16 hasta el 09-10-16. De momento, vamos según lo planificado, incluso algo más adelantados debido a que en la parte de diseño ha habido menos trabajo del que se esperaba.

INFORME DEL MES DE OCTUBRE

A día 9 de noviembre del 2016 el estado del proyecto es el siguiente:

- Primera iteración:
    - Una vez establecido el diseño, hemos implementado el servicio. Al ser un leguaje que ya dominaba, ha ido más rápido de los esperado, y hemos finalizado la implementación en 30h.
    - Una vez terminada la implementación, nos hemos dedicado a realizar una serie de pruebas para comprobar que funciona correctamente. Dedicamos 5h a esto.

- Segunda iteración:
    - Hemos realizado el análisis de la segunda iteración, teniendo en cuenta todo lo que queríamos que cumpliese. Esto nos ha llevado 5h.
    - Hemos comenzado con el diseño de la aplicación móvil, por el momento es bastante sencillo. De momento hemos dedicado 5h.

- Documentación:
    - Todo lo que vamos realizando y probando, se va documentando correctamente, tanto las reuniones como los obstáculos que nos vamos encontrando.

Conclusiones: Este es el segundo informe que abarca el periodo del 10-10-2016 hasta el 09-11-16. La implementación de la primera iteración ha sido un éxito, y se ha terminado antes de lo esperado, esto nos ha hecho volver a definir la duración del proyecto, ya que nos adelantamos alrededor de una semana.

INFORME DEL MES DE NOVIEMBRE

A día 9 de diciembre del 2016 el estado del proyecto es el siguiente:

- Segunda iteración:
    - Hemos concluido con el diseño de la aplicación móvil, aunque nos ha costado concretar el funcionamiento que iba a tener esta, y por lo que nos hemos desviado unas horas de lo previsto. Dedicación, 10h.
    - Después de tener el diseño definitivo, comenzamos la implementación de la aplicación móvil. Hemos dedicado gran parte del tiempo a la comprensión de este tipo de proyectos y como deben desarrollarse correctamente. Nos ha costado dar con la forma correcta de enfocar la implementación de la aplicación, pero ahora vamos por el buen camino. Dedicación, 60h.

- Documentación del proyecto:
    - La documentación va creciendo día a día, anotando cada paso que se realiza. En esta fase no se ha dedicado mucho tiempo, ya que he estado más centrado en la implementación de la aplicación móvil.

Conclusiones: Este es el tercer informe, que abarca el periodo del 10-11-16 hasta el 09-12-16. De momento, vamos retrasados con lo que habíamos vuelto a definir en el segundo informe, pero aun contamos con una semana de ventaja que nos ahorramos con la primera iteración.

INFORME DEL MES DE DICIEMBRE

A día 9 de enero del 2016 el estado del proyecto es el siguiente:

- Segunda iteración:
    - Hemos concluido la implementación de la segunda iteración, y ha llevado más tiempo del esperado. No se ha terminado de dejar la aplicación visualmente como se deseaba, aunque esto no influye en el funcionamiento de esta. Tiempo dedicado, 20h.

- Pruebas y evaluación:
    - Una vez terminada la segunda iteración, es hora de probar que todo funcione en conjunto. El servicio ha sido ubicado en un servidor de Azure, y la aplicación ha sido instalada en dos dispositivos móviles, un Android y un Windows Phone (hasta ahora todas las pruebas se habían realizado en emuladores). Después de unas correcciones y ajustes para las versiones finales, funcionan correctamente.

- Redacción de la memoria:
    - Una vez terminadas las dos aplicaciones, damos por finalizado el proceso de documentación del proyecto y comenzamos con la redacción de la memoria. Siguiendo la estructura que nos aconsejó el directo del proyecto.

Conclusiones: Este es el cuarto informe, que abarca el periodo del 10-12-16 hasta el 09-01-17. La conclusión que saco de este periodo es que, debido a una mala previsión de las horas, ha llevado mucho más tiempo del que se esperaba la segunda iteración. Pero, por otro lado, nos ha salvado el haber terminado antes de tiempo la primera iteración.

INFORME DEL MES DE ENERO

A día 20 de enero del 2016 el estado del proyecto es el siguiente:

- Redacción de la memoria:
    - Seguimos redactando la memoria, hemos terminado los puntos de explicación de las aplicaciones, y ahora estamos redactando la gestión del proyecto, las conclusiones y la bibliografía.

Conclusiones: Este es el quinto informe, que abarca el periodo del 10-01-17 hasta el 20-01-17. La conclusión que saco de este periodo es que, he calculado mal el tiempo de redacción de la memoria, ya que ha sido necesario meter horas extra para la redacción y buena estructuración de esta.

10.2 ANEXO B: MANUAL DE INSTALACIÓN

Servicio REST:

Si se desea instalar el servicio en un equipo local para realizar pruebas, sería necesaria la instalación de Microsoft Internet Information Services, Microsoft SQL Server Managment Studio y Entity Framework. Después habría que crear el modelo de la base de datos con Entity Framework (la base de datos no se entrega porque contiene datos de la empresa, pero bastaría con crear una pequeña tabla como la que se muestra en el apartado 7.1.3). Una vez configurado todo, con iniciar el servicio ya estaría en marcha.

Somos conscientes de que poner en marcha el servicio puede ser un trabajo muy laborioso. Y, probablemente, sin todas las herramientas de las que disponemos en la empresa, sea una tarea un tanto complicada. Por este motivo, la empresa ha puesto a disposición de todos, este mismo servicio en un servidor de Azure. El servicio está en constante ejecución, y no es necesaria su instalación. Para poder acceder al servicio se navega a http://esmartcore.vector-erp.es/, y para poder consultar las llamadas a las funciones a http://esmartcore.vector-erp.es/Help, ahí es posible leer las respuestas que envía el servidor en formato JSON.

Aplicación móvil:

- Android: La aplicación móvil, en el caso de Android, es un archivo ".apk", el cual simplemente hay que copiarlo en la memoria interna del teléfono Android. Previamente, deberemos habilitar la opción de permitir instalar aplicaciones de terceros y, después, simplemente desde el móvil, seleccionando la aplicación, comenzará a instalarse.

- Windows 10: En Windows 10, el propio terminal nos ofrece un sistema de gestión para la instalación de las aplicaciones. Los pasos a seguir son:
    - En primer lugar, hacer clic con el botón derecho sobre el proyecto de Windows y seleccionar **Tienda** > **Crear** paquetes de aplicaciones.

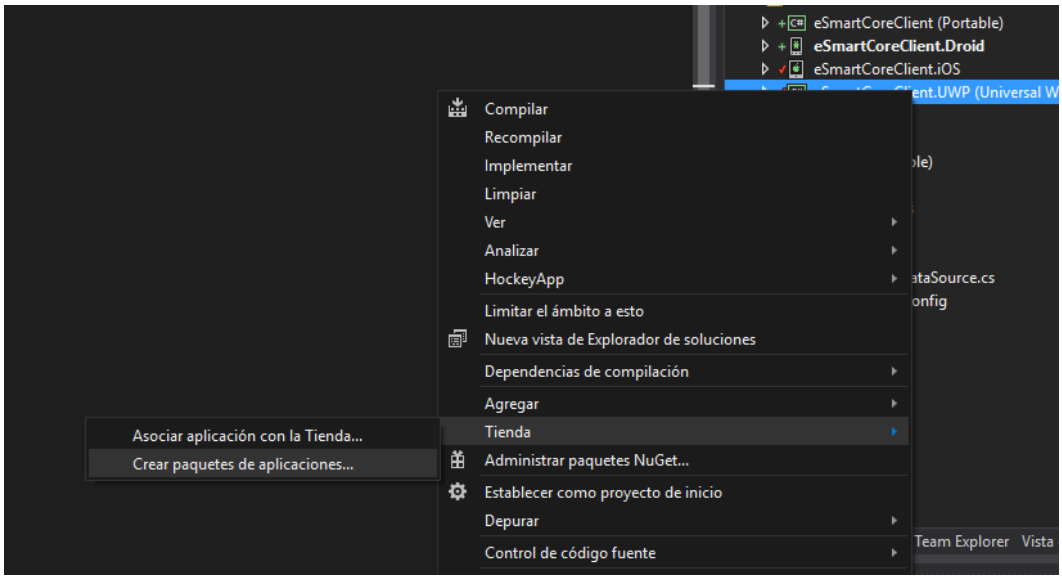


Imagen 10.2.1. Paso 1 manual instalación.

A continuación, seleccionamos la ubicación donde queremos que se guarde la aplicación

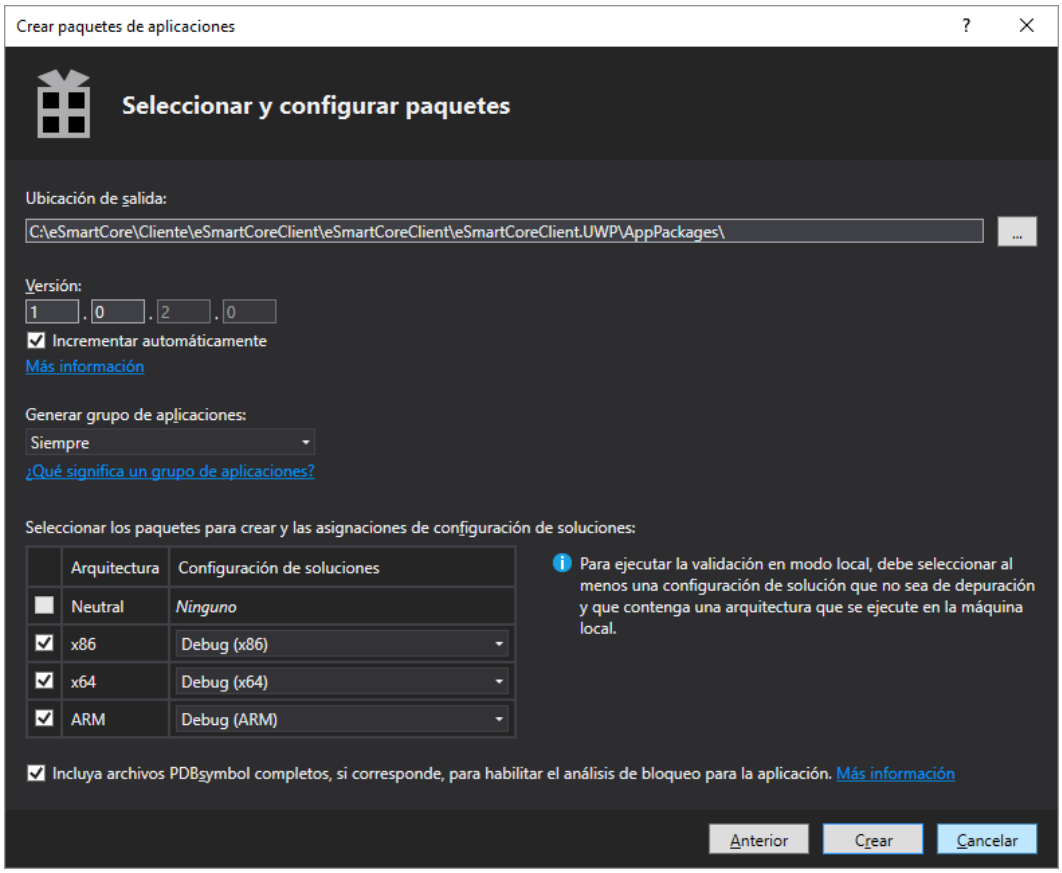


Imagen 10.2.2. Paso 1 manual instalación.

Una vez hecho esto, ya tenemos la aplicación preparada para instalar. Ahora conectándonos por red al terminal, mediante su IP, este nos proporciona el sistema de gestión para instalar aplicaciones.

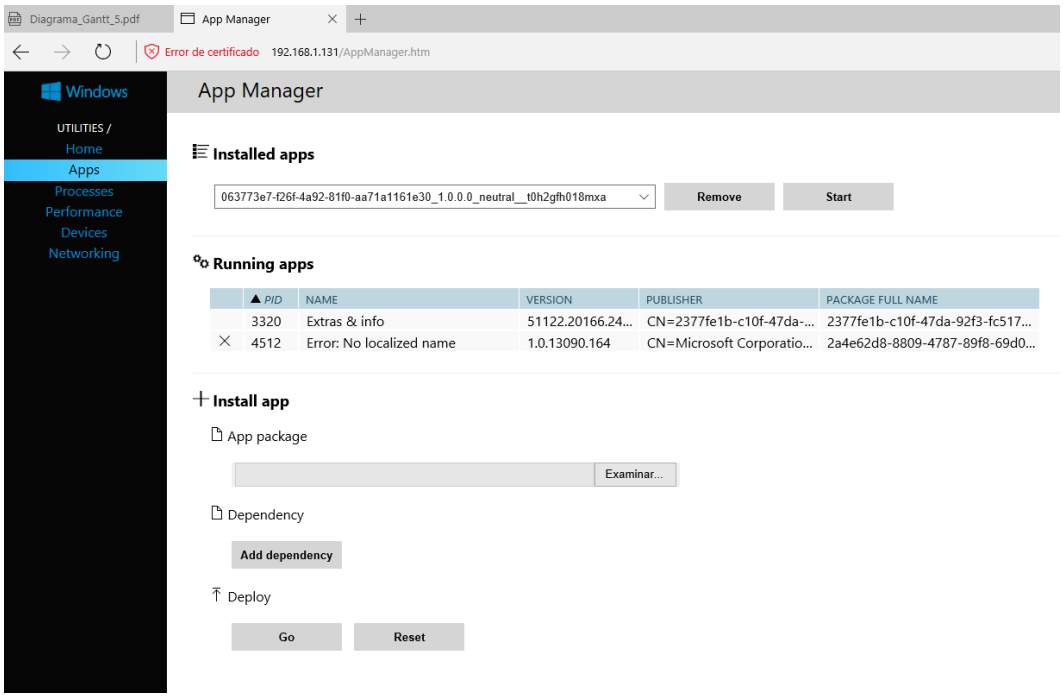


Imagen 10.2.3. Paso 1 manual instalación.

En esta pantalla, en el apartado **App package** debemos seleccionar la aplicación que habíamos guardado anteriormente. En **Dependency** debemos agregar todas las dependencias que nos haya generado el sistema, que se encuentran junto con la aplicación que habíamos guardado.

Una vez hecho esto, en el apartado **Deploy** debemos seleccionar **Go** y comenzará la instalación.

- iOS: Las pruebas con la aplicación de iOS requieren un ordenador Macintosh, por lo que no hemos probado la aplicación en este sistema operativo, y tampoco hemos podido realizar la instalación en ningún iPhone, debido a que no hemos adquirido la licencia de desarrolladores que oferta Apple con un coste de 99€

10.3 ANEXO C: ACTAS DE REUNIÓN

ACTA Nº: 1

LUGAR: Despacho de reuniones de IDS

FECHA: 09-09-16

DURACION DE LA REUNION: 10:30-11:30

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Presentación

TEMAS TRATADOS:

Presentaciones, opiniones del proyecto, dudas generales.

ACUERDOS ALCANZADOS:

Establecemos los medios de comunicación, que serán el correo electrónico y el teléfono, aparte de una reunión diaria.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 2

LUGAR: Despacho de reuniones de IDS

FECHA: 14-09-16

DURACION DE LA REUNION: 10:30-11:15

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Explicación del funcionamiento de la empresa y establecer comienzo del periodo de formación en la tecnología de la empresa.

TEMAS TRATADOS:

Horarios de formación.

ACUERDOS ALCANZADOS:

Se procede a darme unos horarios y días en los cuales me van a impartir los conocimientos que necesitare para el desarrollo del proyecto.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 3

LUGAR: Despacho de José Ángel Vadillo en la Faculta de Informática

FECHA: 15-09-16

DURACION DE LA REUNION: 10:30-11:15

ASISTENTES:

José Ángel Vadillo

Mario Afonso

ORDEN DEL DIA:

Dudas sobre el proyecto para la realización del DOP.

TEMAS TRATADOS:

Especificación del proyecto: metodología y objetivos del proyecto.

Ideas para comenzar con el DOP

ACUERDOS ALCANZADOS:

Entrega de un borrador del DOP.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 4

LUGAR: Despacho de reuniones de IDS

FECHA: 21-09-16

DURACION DE LA REUNION: 10:50-11:50

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Puntos a modificar en el DOP.

TEMAS TRATADOS:

Programas que me tienen que proporcionar para que desarrolle el proyecto, como Visio, Office, etc.

ACUERDOS ALCANZADOS:

Instalar todo lo necesario cuanto antes.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 5

LUGAR: Despacho de reuniones de IDS

FECHA: 20-10-16

DURACION DE LA REUNION: 10:50-11:50

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Evaluación del porcentaje avanzado en la primera iteración y en la memoria

TEMAS TRATADOS:

Se prevé que la primera iteración se terminara a lo largo de la siguiente semana. Y la documentación se está desarrollando correctamente.

ACUERDOS ALCANZADOS:

Modificación de la duración del proyecto, debido a que vamos avanzados.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 6

LUGAR: Despacho de reuniones de IDS

FECHA: 23-11-16

DURACION DE LA REUNION: 10:30-11:50

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Evaluación del porcentaje avanzado en la segunda iteración y en la memoria

TEMAS TRATADOS:

La segunda iteración nos está llevando más tiempo del previsto

La memoria avanza correctamente

ACUERDOS ALCANZADOS:

Modificar el alcance del proyecto de nuevo.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 7

LUGAR: Despacho de reuniones de IDS

FECHA: 21-12-16

DURACION DE LA REUNION: 10:30-11:30

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Segunda iteración, pruebas y memoria

TEMAS TRATADOS:

Se estima que la segunda iteración será concluida a lo largo de los próximos días.

Se habla de que pruebas realizaremos para comprobar la funcionalidad de la aplicación.

La memoria avanza correctamente.

ACUERDOS ALCANZADOS:

Diseñar las pruebas.

PROXIMA REUNION:

Se concretará en los próximos días.

ACTA Nº: 8

LUGAR: Despacho de reuniones de IDS

FECHA: 16-01-16

DURACION DE LA REUNION: 10:50-11:50

ASISTENTES:

Edorta Guisasola

Mario Afonso

ORDEN DEL DIA:

Modificación de la estructura de la memoria.

TEMAS TRATADOS:

Tras enviar unas versiones al director del proyecto, nos recomienda seguir una estructura diferente.

ACUERDOS ALCANZADOS:

Modificar de acuerdo a las indicaciones del director del proyecto la memoria.

PROXIMA REUNION:

Se concretará en los próximos días.

10.4 ANEXO D: CONCEPTOS TEÓRICOS

Aplicación cliente: Es la entidad por medio de la cual un usuario solicita un servicio. Se encarga básicamente de presentar los datos y/o información al usuario en un ambiente gráfico.

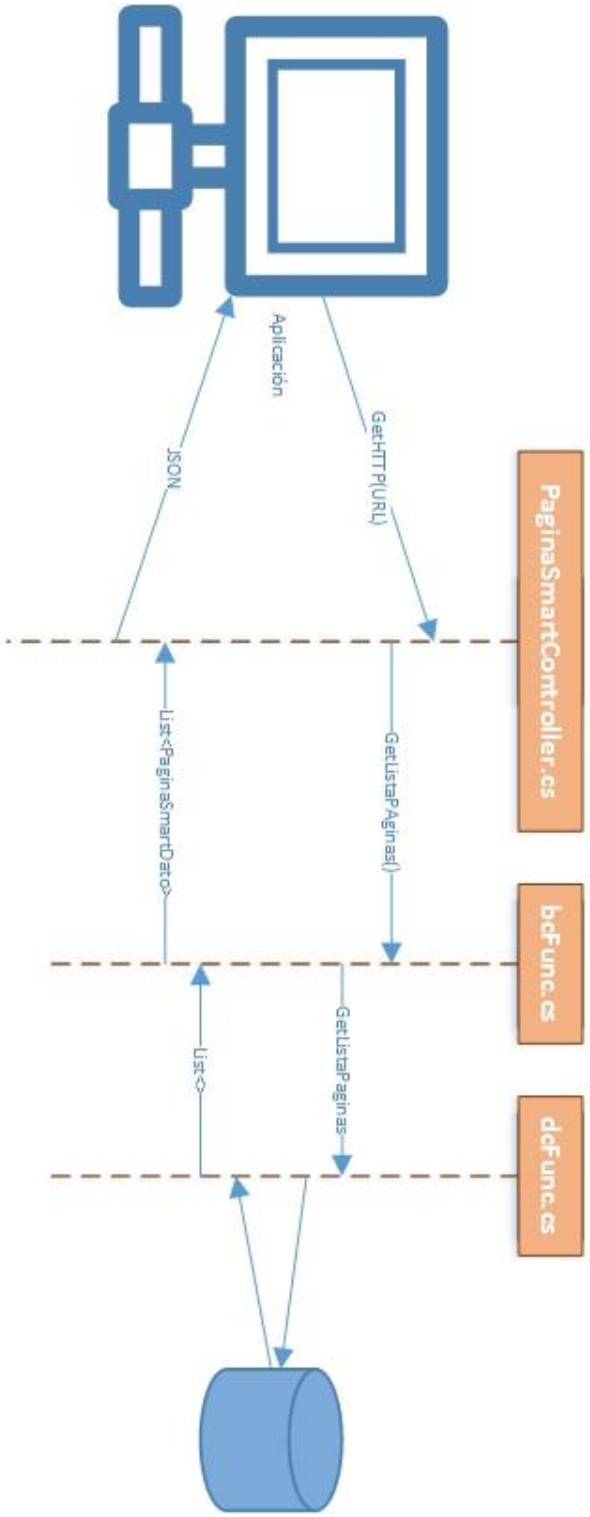
Servicio REST: Es la entidad que se encuentra en Azure, que provee y devuelve resultados, ejecuta el procesamiento de datos y manejo de la información o recursos. En el servicio se realiza la parte destinada a recibir las solicitudes del cliente y donde se ejecutan los procesos.

Xamarin, Syncfusion: Son herramientas de desarrollo para el entorno Visual Studio, que nos dan facilidades para programar la parte del cliente. Nos ayudan a hacer la parte visual y la posibilidad que tiene nuestra aplicación de ejecutarse en cualquier dispositivo.

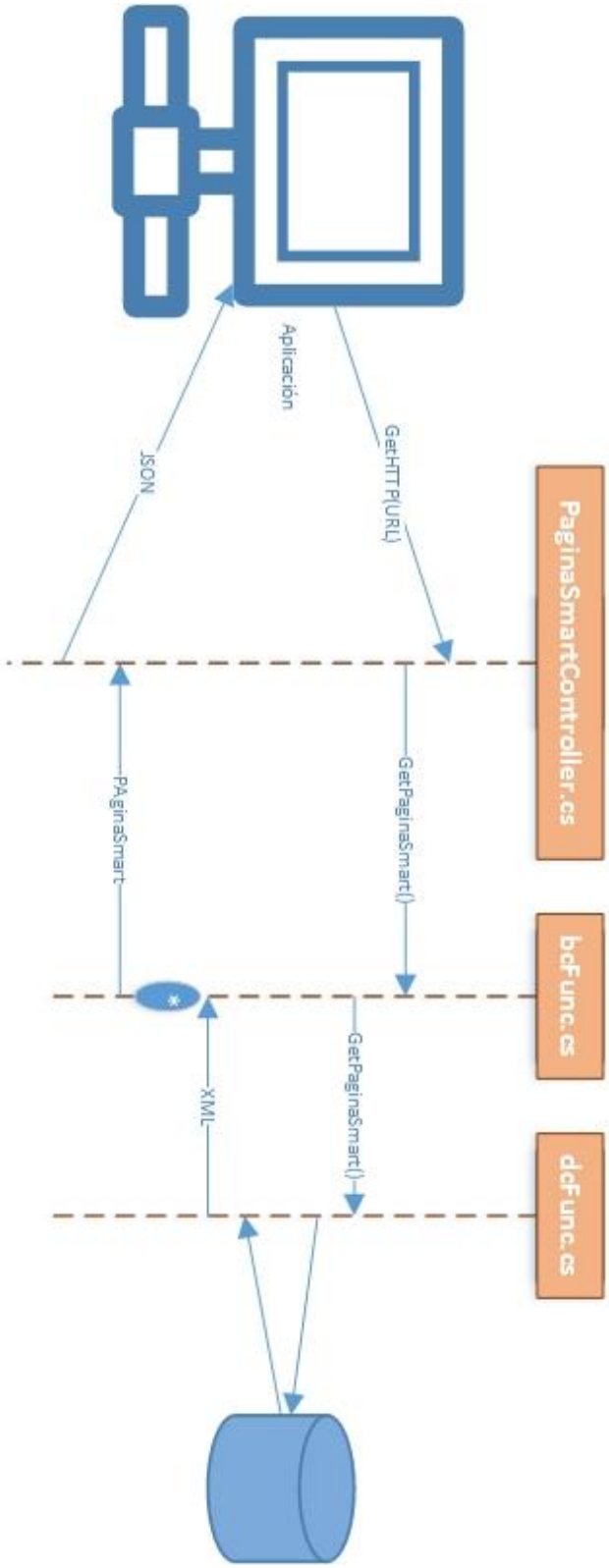
Visual Studio: Es el entorno de trabajo que vamos a utilizar, es donde programaremos tanto el servicio REST como la aplicación móvil.

10.5 ANEXO E: IMÁGENES AMPLIADAS

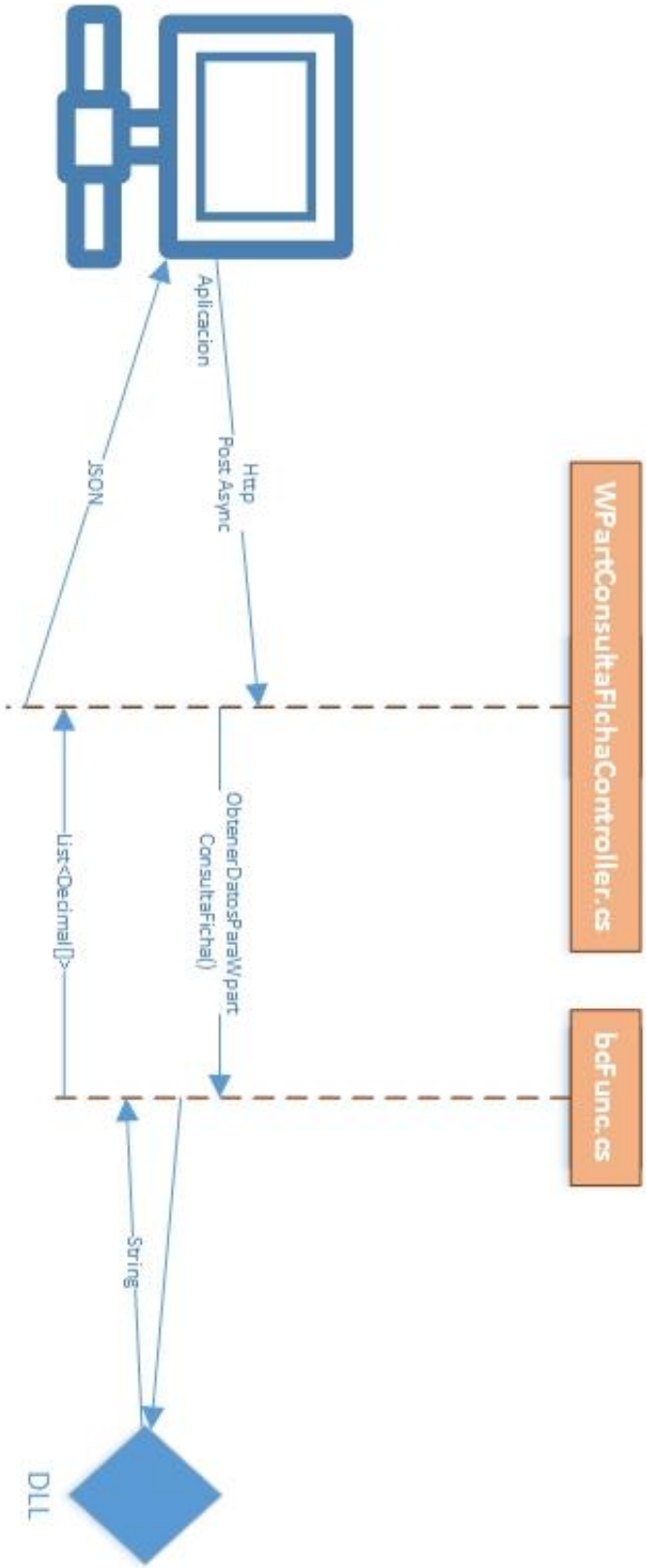
10.5.1 Imagen Ampliada: Servicio, Petición de la lista de paginas



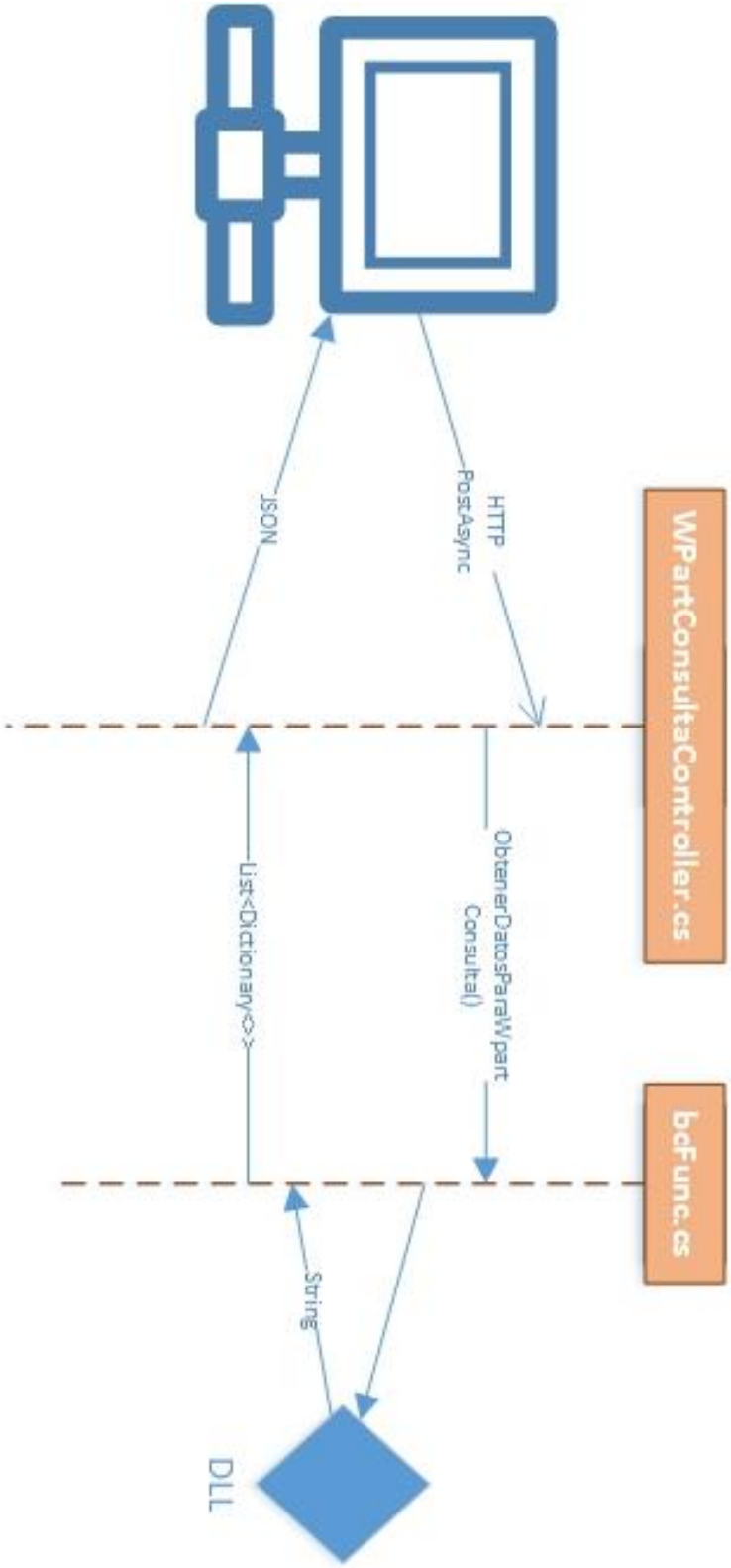
10.5.2 Imagen Ampliada: Servicio, Petición de una página Smart



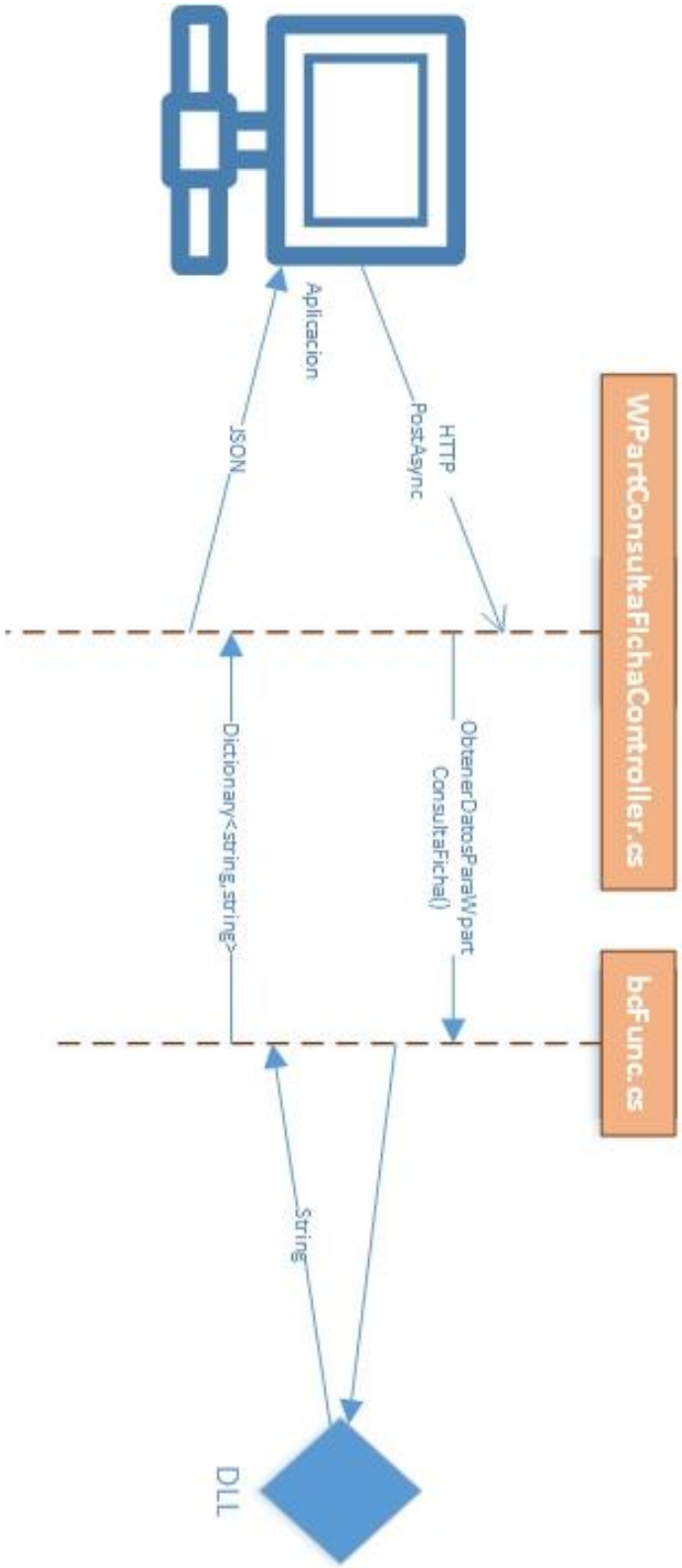
10.5.3 Imagen Ampliada: Servicio, Petición de los datos de un grafico



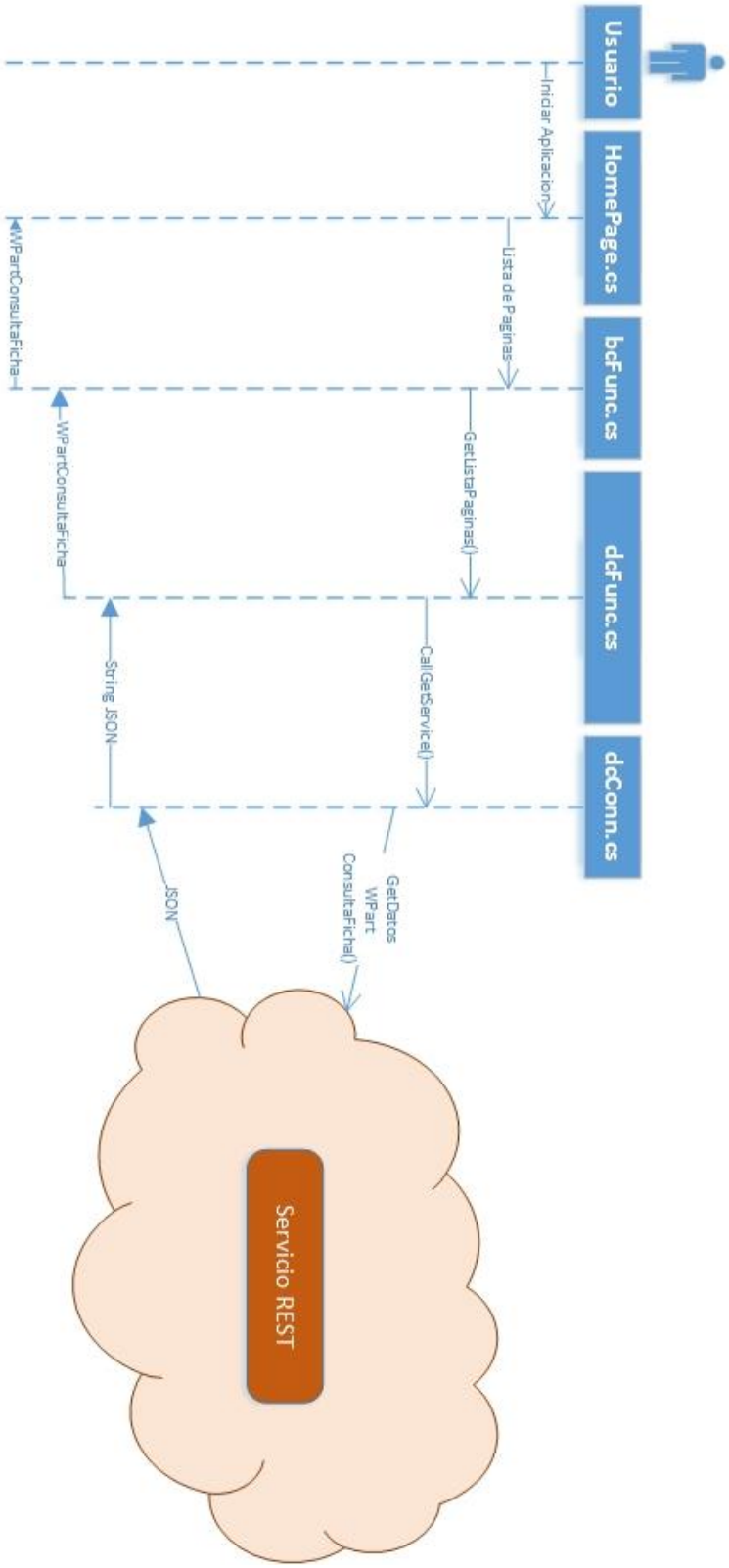
10.5.4 Imagen Ampliada: Servicio, Petición de los datos de una consulta



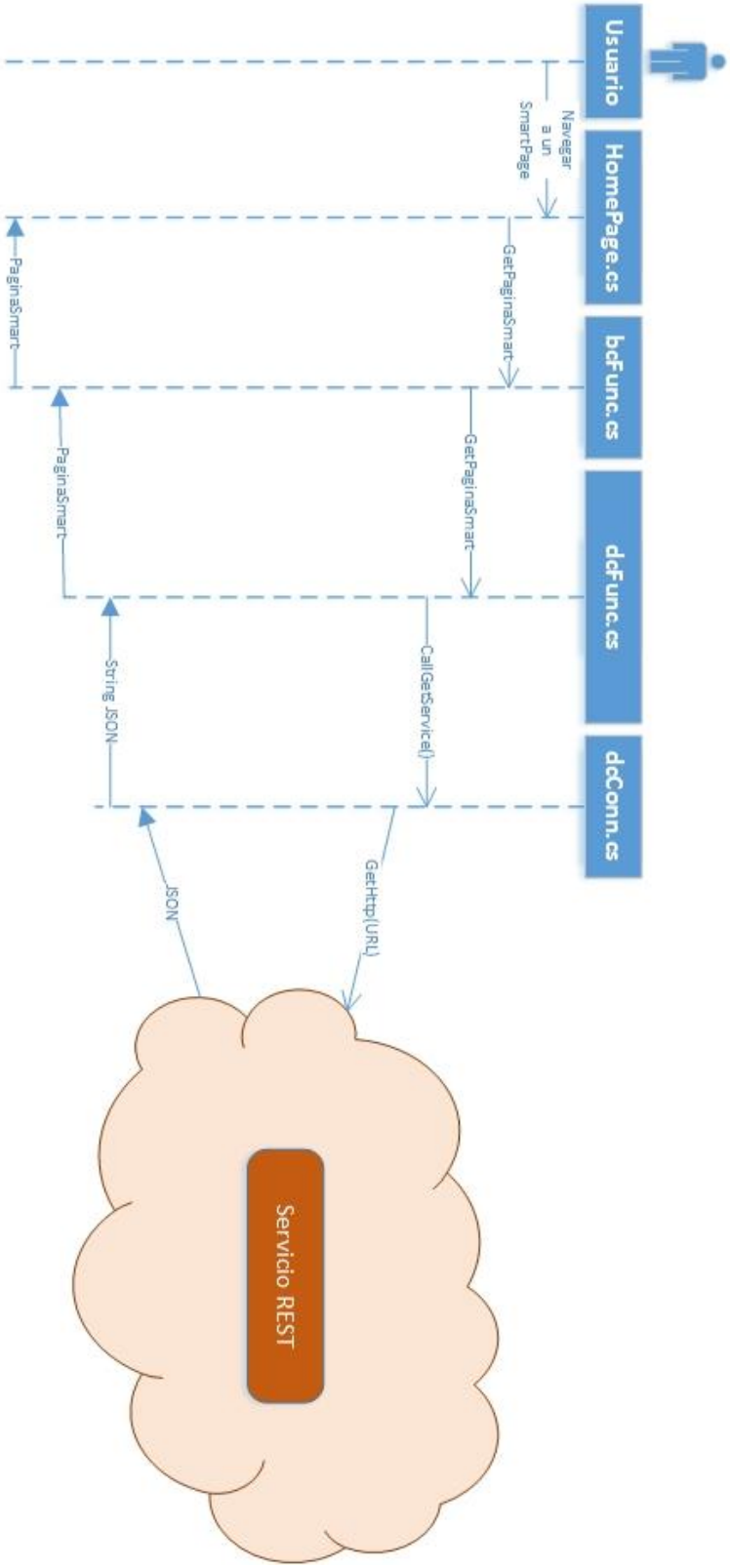
10.5.5 Imagen Ampliada: Servicio, petición de los datos de una consulta tipo ficha



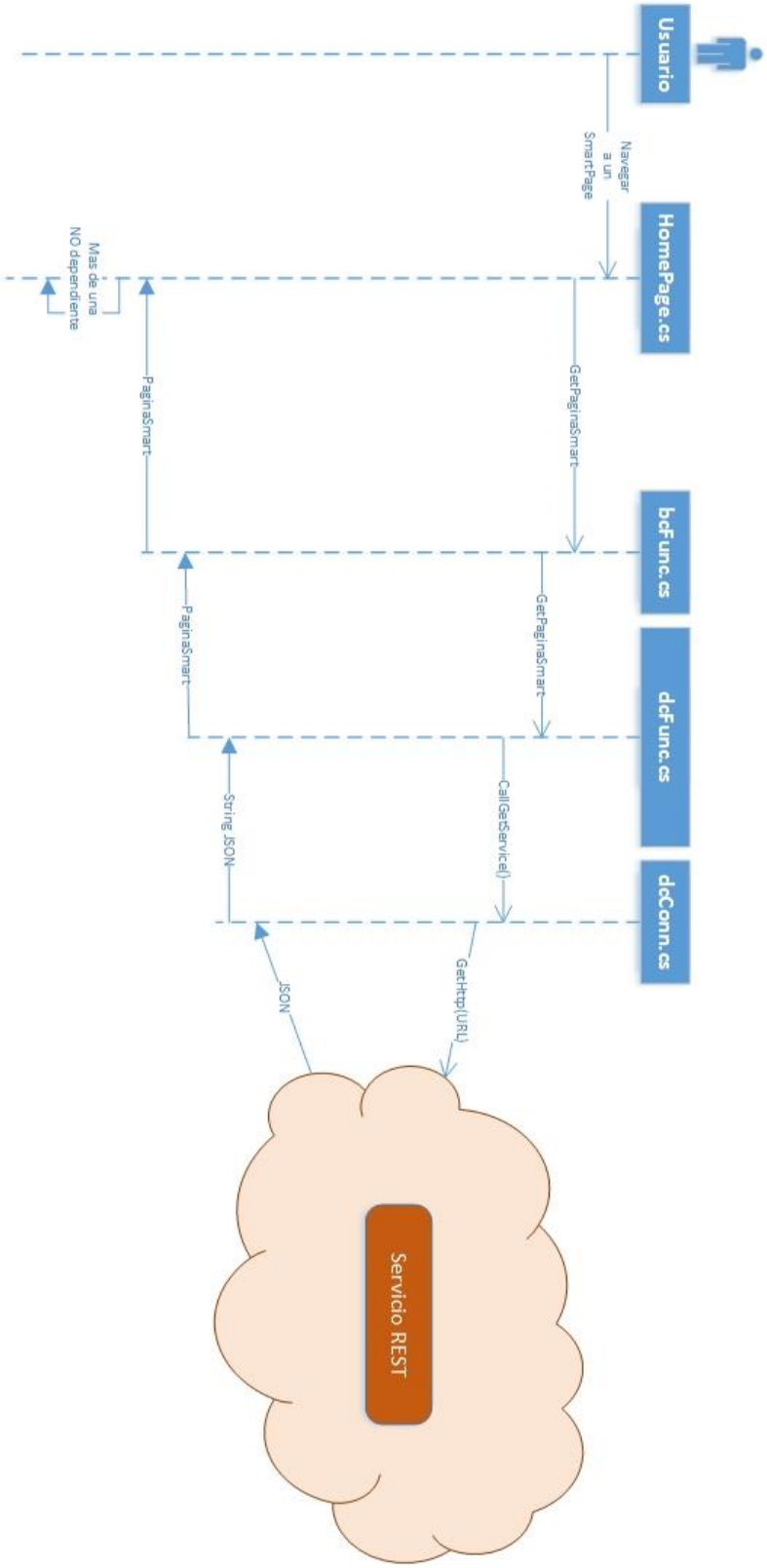
10.5.6 Imagen Ampliada: Cliente, entrada en la aplicación



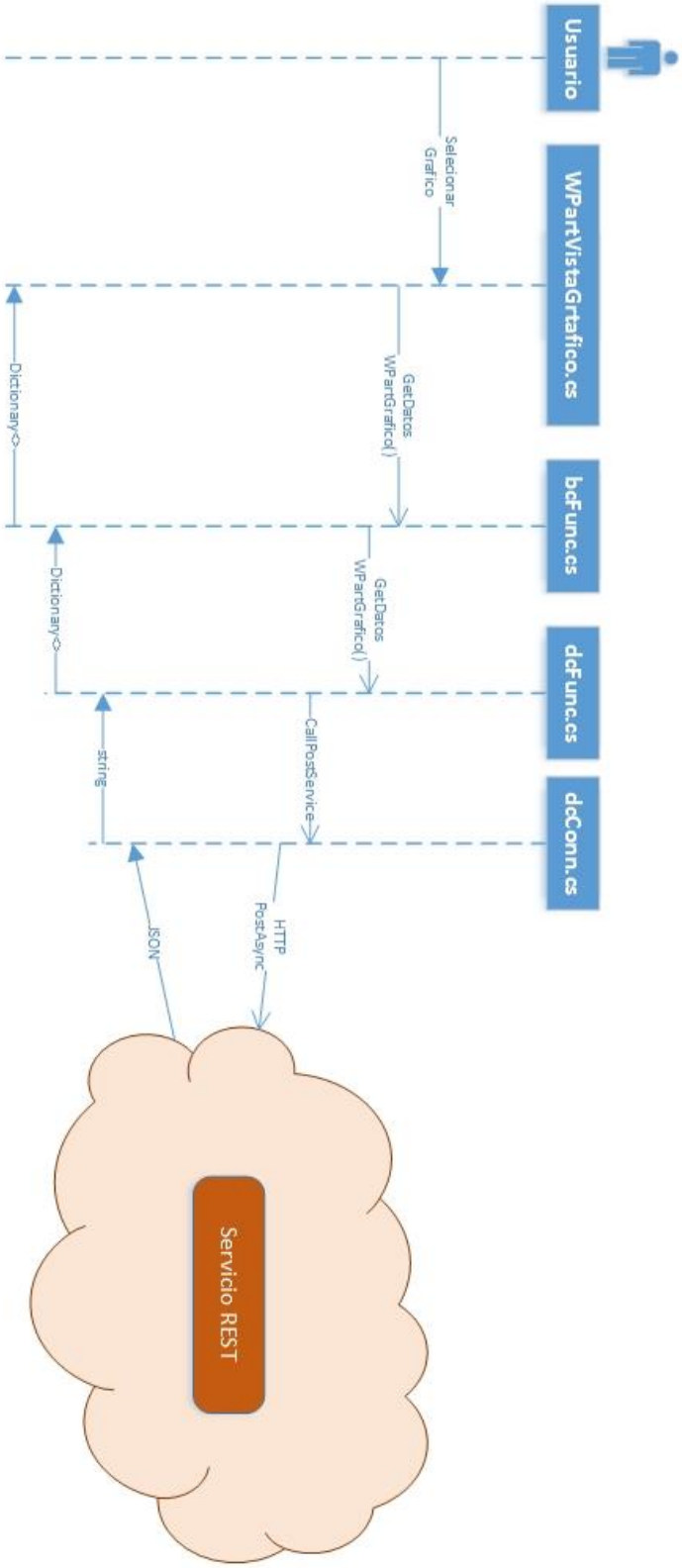
10.5.7 Imagen Ampliada: Cliente, seleccionar una página, caso 1



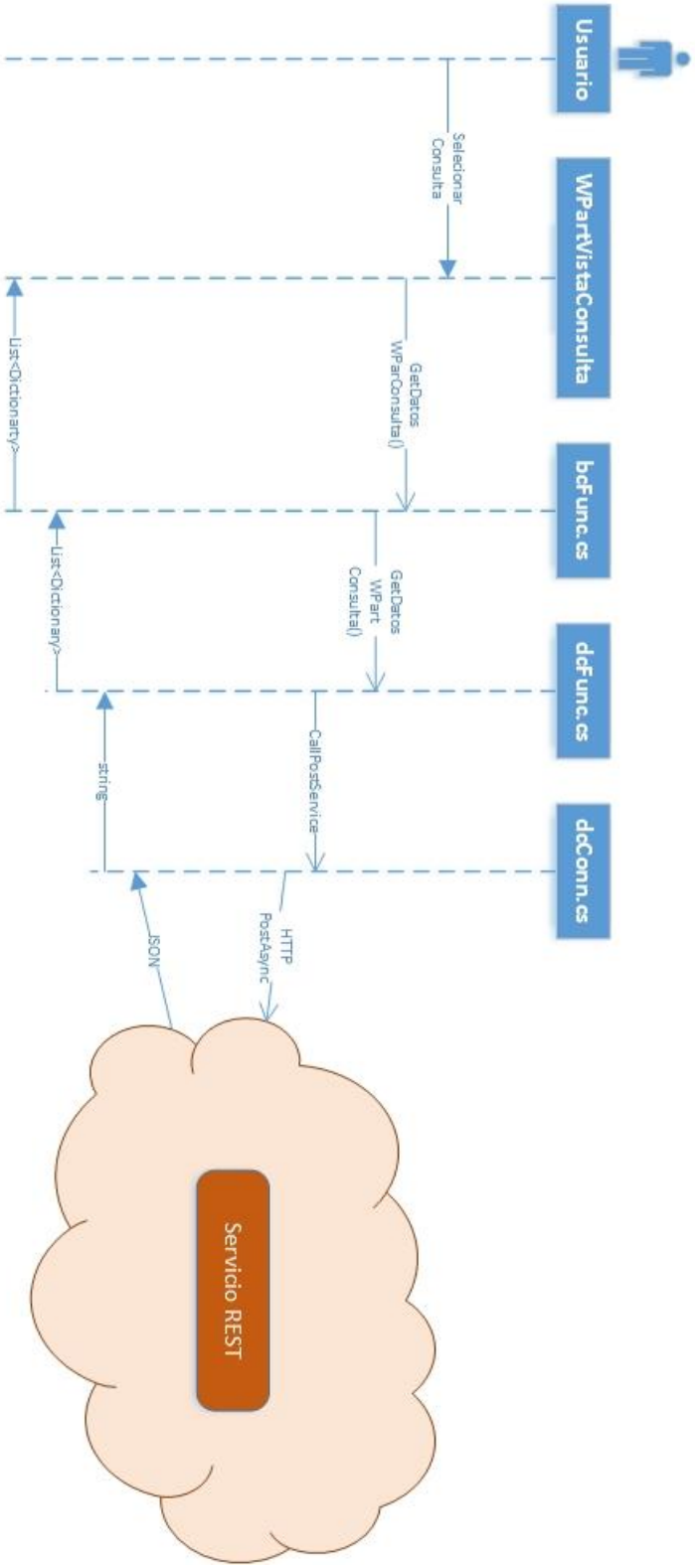
10.5.8 Imagen Ampliada: Cliente, seleccionar una página, caso 2



10.5.9 Imagen Ampliada: Cliente, seleccionar un grafico



10.5.10Imagen Ampliada: Cliente, seleccionar una consulta de tipo lista



10.5.11Imagen Ampliada: Cliente, seleccionar una consulta de tipo ficha